TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

# Machine Learning Methoden als Recommender Systems zur Prognose von Cross-Selling Potentialen: Eine Fallstudie in der Versicherungsbranche

ausgeführt am

Institut für
Stochastik und Wirtschaftsmathematik
TU Wien

unter der Anleitung von

**Univ. Prof. Dipl.-Ing. Dr. techn. Stefan Gerhold**

durch

**Armin Malleier, BSc**
Matrikelnummer: 01426290

Wien, am 02.12.2024

[redacted] [redacted]

(Unterschrift Verfasser) (Unterschrift Betreuer)

# Kurzfassung

Eine Verkaufsstrategie, die darauf abzielt, Kunden zum Kauf weiterer, passender Produkte oder Dienstleistungen zu bewegen, wird als Cross-Selling bezeichnet. Sie trägt entscheidend dazu bei, den Wert eines Kunden zu steigern und seine Zufriedenheit durch maßgeschneiderte Produktvorschläge zu erhöhen. Empfehlungssysteme sind hierbei wichtige Werkzeuge, da sie das Verhalten der Nutzer analysieren und gezielte Produktempfehlungen geben. Herkömmliche Methoden stoßen oft auf Herausforderungen wie unzureichende Daten und Probleme eine Empfehlung bei neuen Kunden abzugeben. Um diese Schwierigkeiten zu überwinden, kommen fortschrittliche Data-Mining und Machine Learning Methoden zum Einsatz. Maschinelles Lernen bringt in der Anwendung einige Vorteile: Es erfasst komplexe, nicht-lineare Zusammenhänge in den Daten und verbessert die Genauigkeit und Anpassungsfähigkeit der Vorhersagen. Durch den Einsatz maschinellen Lernens in Empfehlungssystemen für die Cross-Selling-Erkennung können Unternehmen Kundendaten und Transaktionshistorien nutzen, um relevante Produktbeziehungen zu identifizieren. Algorithmen wie Gradient Boosting und Deep Learning erlauben es, Empfehlungen laufend anzupassen und zu verbessern, sobald neue Daten verfügbar sind. Diese Arbeit untersucht, wie maschinelles Lernen dazu beiträgt, Cross-Selling-Strategien zu optimieren, um die Kundenbindung zu erhöhen und den Umsatz durch präzise Vorhersagemodelle zu steigern. Dazu wurden die Modelle mit den Daten des Versicherungsunternehmens Allianz Österreich trainiert. Dabei erzielten Random Forests und Catboost die höchsten F1-Scores und wurden als bestes Modell gekürt. Der Ansatz Risikoprofile zu erstellen, anstatt die Versicherungsprodukte als Zielklasse auszuwählen, war mitunter einer der entscheidenden Schritte dieser Arbeit. Zudem machte sich die unterschiedliche Verwendung verschiedener Merkmale als Trainingsdatensatz signifikant bemerkbar.

# Abstract

Cross-selling is a sales approach aimed at encouraging customers to buy additional, related products or services. This strategy helps increase the overall value of a customer and boosts their satisfaction through personalized product recommendations. Recommendation systems play an essential role in this process by analyzing user behavior to deliver targeted product suggestions.

Traditional methods can face obstacles, such as having limited data or difficulties in making recommendations for new customers. To overcome these issues, advanced data mining and machine learning methods are applied. Machine learning stands out for its ability to understand complex, non-linear patterns in data and make predictions that are more accurate and adaptable.

When machine learning is used in recommendation systems to identify cross-selling opportunities, companies can leverage customer data and transaction histories to spot relevant product connections. Algorithms like gradient boosting and deep learning enable continuous updates and enhancements to recommendations as new data is collected. This thesis explores how machine learning improves cross-selling strategies, enhancing customer loyalty and increasing sales with precise predictive models.

In this study, data from the insurance company Allianz Austria was used to train various models. In the process the Random Forests and Catboost achieved the highest F1 score and were determined as the best. An important part of this work was creating risk profiles instead of directly targeting specific insurance products. Additionally, the choice and use of different features for the training dataset played a major role in the outcomes.

# Danksagung

Zu Beginn möchte ich meinen Dank all denjenigen aussprechen, die mich während der Studienzeit unterstützt haben. Mein besonderer Dank gilt meinem Universitätsbetreuer Prof. Dr. Stefan Gerhold, sowie meinem Betreuer in der Allianz Elementar Vers. AG Österreich, Patrick Hippman, Phd für ihre wertvolle Unterstützung. Dank gebührt auch den Kolleg:innen der Allianz Österreich, die mir bei der Beschaffung und Erklärung der Daten zur Seite gestanden haben. Einen besonderen Dank möchte ich auch an meine Familie, insbesondere meine Eltern richten, ohne die mein Studium nicht möglich gewesen wäre. Ihre Ermutigung und Verständnis waren für mich von unschätzbaren Wert. Ein herzliches Dankeschön gilt auch an meine Freundin und Freunde, die mir bei dieser Arbeit in den unterschiedlichsten Art und Weisen geholfen haben.

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 02.12.2024

Armin Malleier

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| AUC | Area Under the Curve |
| ANN | Artificial Neural Network |
| ANOVA | Analysis of Variance |
| APA | Acquisition Pattern Analysis |
| Bzw. | beziehungsweise |
| ca. | circa |
| CatB | Catboost |
| CPU | Central Processing Unit, Prozessor |
| CRM | Customer Relationship Management |
| CF | Collaborative Filtering |
| d.h. | das heißt |
| DT | Decision Tree |
| EFB | Exclusive Feature Bundling |
| etc. | et cetera |
| FNN | Feedforward Neural Network |
| FS | Feature Selection |
| GBDT | Gradient Boosted Decision Trees |
| GFB | Geschäftsfallbearbeitungssystem |
| GOSS | Gradient-based One-side Sampling |
| KFZ | Kraftfahrzeug |
| KDD | Knowledge Discovery in Databases |
| KPI | Key Point Indicator |
| k-NN | k-nearest Neighbors |
| LASSO | Least Absolute Shrinkage and Selection Operator |
| LDA | Linear Discriminant Analysis |
| LR | Logistische Regression |
| LGBM | LightGBM |
| ML | Machine Learning |
| MV | Missing Values |
| NB | Naive Bayes |
| NN | Neural Network |
| o.ä. | oder ähnlich |
| OOB | Out of Bag |
| PCA | Principal Component Analysis |
| QDA | Quadratic Discriminant Analysis |
| RAM | Random-Access Memory |
| ReLU | Rectified Linear Unit |
| ROC | Receiver Operating Characteristic |

| | |
|---|---|
| SMOTE | Synthetic Minority Oversampling Technique |
| u.a. | unter anderem |
| VN | Versicherungsnehmer |
| VU | Versicherungsunternehmen |
| vgl. | vergleiche |
| XGB | XGBoost |
| z.B. | zum Beispiel |

# 1. Einleitung

Cross-Selling ist eine Verkaufsstrategie, die darauf abzielt, bestehende Kunden gezielt dazu zu motivieren, weitere Produkte oder Dienstleistungen zusätzlich zu erwerben. Diese Strategie hat sich im Bereich des *Customer Relationship Management* aus mehreren Gründen als lohnenswert erwiesen. Inzwischen wurde gezeigt, dass ein überhäufen von Kaufsangeboten oft zu einer entgegengesetzten Konsequenz führt und Kunden dadurch verärgert werden können [1]. Um ein gezieltes Angebot angeben zu können, bedarf es einer Empfehlung zum Kauf durch eine Prognose des Kundenverhaltens. Eine solche Methode wird als *Recommender System* bezeichnet. Aus den verschiedenen Methoden für solche Systeme wird für diese Arbeit ein modellbasierter *Collaborative Filtering* Ansatz verwendet [2]. Das von den Modellen zu trainierende Ziel wird als eine Gruppierung mehrerer Einzelrisken auf verschiedene Hauptrisikosegmente der Allianz Österreich definiert. Dabei wird das Kaufverhalten aller VN des Unternehmens ein Jahr lang beobachtet und auf dieser Verteilung eine Prognose für das nächste Jahr kalkuliert. Die verwendeten personenbezogenen Daten werden in Kooperation mit der Allianz Österreich erstellt und das Ergebnis ist als Fallstudie für dieses Versicherungsunternehmen zu verstehen.

Damit soll folgende Forschungsfrage untersucht werden: Mit welcher Methode gelingt es am besten, einen Privatkunden der Allianz Elementar Vers. AG Österreich ein zusätzliches Risikosegment vorzuschlagen, um diesen damit ein, mit hoher Wahrscheinlichkeit, zusätzliches Versicherungsprodukt zu verkaufen?

Um das beste Modell aus einem möglichst großes Spektrum an verschiedenen Methoden auswählen zu können, wurden für diese Arbeit zehn verschiedene *Machine Learning* Algorithmen auf fünf verschiedene Szenarien trainiert und getestet. Die zu beobachtenden Szenarien wurden aufgrund ihrer hohen Signifikanz (im Vergleich zu anderen Szenarien) ausgewählt.

Diese Diplomarbeit wurde in den folgenden Kapiteln aufgeteilt. Kapitel 2 zeigt einen Literaturüberblick von Cross-Selling bis hin zu dem Zusammenhang von *Recommender Systems* und *Machine Learning*. Es wird auch auf verschiedene Studien zu Cross-Selling in der Versicherungsbranche eingegangen und deren Ergebnisse für diese Arbeit berücksichtigt. In Kapitel 3 werden die verschiedenen *Machine Learning* Modelle hergeleitet und deren teils einschneidenden Annahmen erklärt. Die Modelle werden dabei neutral beschrieben und deren Algorithmen können auch für andere Zwecke genutzt werden. Das Kapitel 4 beschreibt die verwendeten Daten und aus welchen Systemen innerhalb der Allianz diese zusammengetragen wurden. Weiters werden einige Methoden zur Analyse der Datengrundmenge vorgestellt und in einem weiteren Schritt gezeigt, wie diese verarbeitet werden sollen, um in die *Machine Learning* Algorithmen eingespeist werden zu können. Dabei wird auch eine wichtige Methode der Merkmalsauswahl erklärt, diese bildet die Grundlage der fünf verschiedenen Szenarien welche für jedes Modell trainiert wurden. Damit ist der theoretische Part der Arbeit vollendet und Kapitel 5 führt den Prozess der praktischen Anwendung

in näherer Betrachtung aus. Schlussendlich werden in Kapitel 6 die Resultate der Modelle auf den verschiedenen Szenarien gezeigt und abschließend in Kapitel 7 noch einmal zusammengefasst und ein Fazit gezogen.

# 2. Literaturüberblick

## 2.1. Cross-Selling

In der zunehmend wettbewerbsintensiven Landschaft der Finanz- und Dienstleistungsmärkte ist die Gewinnung neuer Kunden oft nur möglich, wenn diese von Konkurrenten abgeworben werden. Zudem neigen viele dieser neu geworbenen Kunden dazu, wieder zu wechseln, sobald sie ein besseres Konkurrenzangebot erhalten. Diese Kundenabwanderung führt für Unternehmen zu hohen Kosten und zu einem unsicheren bzw. nicht nachhaltigen Ertrag. Um aus diesem Kreislauf auszubrechen, bauen Unternehmen immer mehr auf stärkere Kundenbeziehungen. Das sogenannte *Customer Relationship Management* (CRM) ist ein unglaublich schnell wachsender Markt, welcher aus Management- und Marketingbereichen nicht mehr wegzudenken ist. Die weltweiten Investitionen für CRM belief sich 1999 auf 3,3 Milliarden Dollar und wuchsen bis 2022 auf satte 58,82 Milliarden Dollar an. Das entspricht einem jährlichen Wachstum von 13,34% und soll Berichten zu folge bis 2030 mit ähnlichen Wachstum weiter steigen [3]. Aktuelle Trends wie die Benutzung von Künstlicher Intelligenz, Automatisierung und das Wachstum von gespeicherten Kundendaten sind die wichtigsten Faktoren, die das Marktwachstum vorantreiben. Es ist also für Unternehmen unabdingbar geworden, eigene Strategien zur Stärkung der Kundenbeziehungen zu entwickeln. Solche sind oft datenbankgestützte Modelle zur Förderung zusätzlicher Verkäufe durch Direktmarketing-Strategien. Direktmarketing gilt als Strategie, die den Fokus auf direkten, persönlichen Kontakt legt, um Dienstleistungen ohne Zwischenhändler an die Kunden zu liefern.
Cross-Selling hat sich als wertvolle Strategie im Bereich des CRM erwiesen. Cross-Selling definiert sich, als der Verkauf zusätzlicher, verwandter Artikel, Produkte oder Dienstleistungen an Bestandskunden, sprich Kunden die bereits davor schon eine Dienstleistung in Anspruch genommen haben. Es gibt einige Gründe dafür, dass sich diese Strategie so stark durchgesetzt hat. Erstens wird angenommen, dass die Betreuung eines bestehenden Kunden nur einen Bruchteil von dem kostet, was für die Anwerbung eines neuen Kunden erforderlich wäre. Anders gesagt ist das ein Mittel zur Kostenoptimierung. Zweitens sind die Rücklaufquoten bei Cross-Selling deutlich höher als bei Kaltakquisen. Drittens wird durch zusätzlichen Verkäufen die Kundenbeziehung erweitert und damit steigt auch das Wissen was dem Kunden mehr zuspricht. Viertens steigen für den Kunden die physischen und psychischen Kosten, welche mit einem möglichen Wechsel zur Konkurrenz verbunden wären. Fünftens erhält das Unternehmen zusätzliche Informationen des Kunden, wenn sich diese über verschiedene Produkte und Dienstleistungen an das Unternehmen wenden. Dadurch kann das Unternehmen seine Marketingstrategien effektiver gestalten und Cross-Selling noch gezielter durchführen [1].
Es reicht jedoch nicht einfach jeden Kunden alle zusätzlichen Produkte vorzusetzen. Übereifrige Cross-Selling Bemühungen können durchaus auch den umgekehrten Effekt bewirken,

wie z.B. die Verärgerung oder Desensibilisierung auf künftige Cross-Selling Versuche (Mails landen im Spam-Ordner). Auch die Umsetzung der Cross-Selling Strategie kann für Unternehmen Vor- und Nachteile haben. Es hat sich bewährt, dass Cross-Selling deutlicher effektiver ist, wenn der Kunde selbst den Kontakt sucht als umgekehrt. Das lässt sich dadurch erklären, dass der Kunde bei Problemen oder Informationen das Unternehmen von sich aus aufsucht und wenn das Problem direkt gelöst werden konnte, fällt die Einstellung vom Kunden zum Unternehmen deutlich positiver aus als wie wenn der Kunde zu einer beliebigen Zeit vom Unternehmen aus kontaktiert wird.
Im Allgemeinen kann man Cross-Selling in zwei Hauptkategorien von Analysetools unterteilen: *Acquisition Pattern Analysis* (APA) und *Collaborative Filtering* (CF). Beim APA geht es darum, den nächst logischen Erwerb von Produkten und/oder Dienstleistungen zu ermitteln. CF hingegen untersucht Muster und Verhaltensweisen zwischen Käufen von verschiedenen aber ähnlichen Kunden und probiert daraus neue Produkte vorzuschlagen.
Diese Arten von Analysetools sind als Hilfsmittel zur Umsetzung von Cross-Selling Strategien im Online-Handel wie z.B. Amazon und auch bei Streaming-Anbieter wie Netflix nicht mehr wegzudenken. Bei Finanzdienstleistungen und -produkten ist die Umsetzung von Cross-Selling ein langsamer Prozess, es gibt nur wenige Studien die sich explizit dem Cross-Selling von Versicherungsprodukten widmen. Dabei ist gerade die Benutzung von CRM-Systemen in so stark umkämpften und gesättigten Märkten extrem wichtig. Versicherungen erfüllen zudem auch alle Kriterien um die oben gennanten Analysetools für Cross-Selling umzusetzen: Kundendatenbanken mit detaillierten Informationen über Kaufverhalten und demographischen Daten, unternehmensweite Data Warehouses für eine ganzheitliche Sicht auf jeden Kunden über alle Kontaktwege und Anwendungen für das Kontaktmanagement (z.B. Leads-Tools) [1].

## 2.2. Data Mining

Es ist bekanntlich nichts Neues, dass es heutzutage in zahlreichen Bereichen zu einer raschen und beträchtlichen Ansammlung von Daten kommt. Die Theorie dahinter, welche sich damit beschäftigt, wertvolle Informationen aus den ständig wachsenden Datenmengen zu extrahieren und daraus hilfreiche computergestützte Werkzeuge zu entwickeln, nennt sich *Knowledge Discovery in Databases* (KDD). Im Grunde geht es bei KDD darum Methoden und Techniken zu entwickeln, welche sich im Mittelpunkt spezifische Data-Mining-Methoden zu Nutze machen. Der Begriff *Data Mining* ist somit, streng genommen, nur ein Teil des KDD. In der Praxis wird er jedoch als Synonym dafür benutzt (vgl. Abbildung 2.1 )

Das für die Umsetzung des Cross-Sellings übergeordnete Hauptziel des Data Minings ist die Schätzung der Vorhersage eines Kaufes von einem spezifischen Kunden, unter der Berücksichtigung des vergangenen Kaufverhalten. Diese Art von Schätzung wird auch als Klassifikationsverfahren bezeichnet. Die Einteilung der Klassen sind in der folgenden Arbeit als die verschiedenen Risikosegmente des Versicherungsunternehmen Allianz Elementar AG zu verstehen. Dieser Ansatz hat den Vorteil, dass sich im Gegensatz zur Klassifizierung von Produkten, die Risikosegmente über die Jahre nicht ändern. Versicherungsprodukte haben in der Praxis immer wieder neue Einteilungen, Klauseln und können auch verschiedene

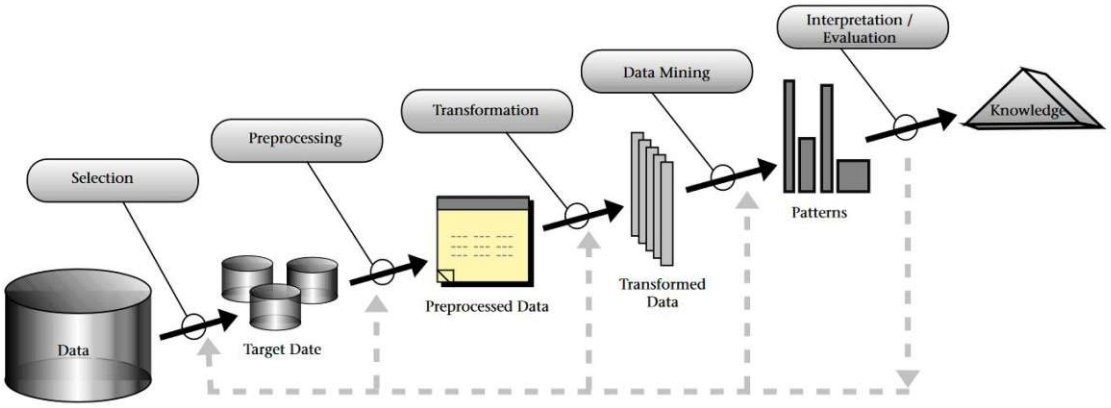


Abbildung 2.1.: Eine Übersicht der einzelnen Schritte des KDD Prozesses [4]

Risiken versichern. Die zu versichernden Risiken bleiben jedoch über die Jahre konstant und sind somit vergleichbarer und besser zum Data Mining geeignet. Die typischen Methoden der Klassifizierung sind die *logistische Regression* (LR), *Decision-Trees* (DT), *Artificial Neural Networks* (ANN) und *Naive Bayes* (NB). Diese Methoden sind oft auch als *Machine Learning Algorithmen* (ML) bekannt.
Data Mining und Machine Learning werden häufig im Sprachgebrauch gleichgesetzt, da sie eine methodische Ähnlichkeit aufweisen. Im Grunde kann man die Verwendung mathematischer und statistischer Techniken zur Verarbeitung gesammelter und aufarbeitenden Daten zur Klassifizierung oder ähnlichem als Data Mining definieren. Der Bereich des ML geht noch einen Schritt weiter, er umfasst auch die Vorhersage künftiger Ereignisse bzw. Käufe und entwickelt sich ständig weiter. Dieses Dazulernen durch neue Informationen ist der Grund warum ML als Untergruppe von künstlicher Intelligenz angesehen wird.
Die Kundeninformationen und deren Verhalten sind in der Versicherungsbranche oft nicht leicht zu modellieren und es kann daraus kein genaues System zur Prognose von spezifischen Kaufverhalten hergeleitet werden. Da aber genügend Daten in den Datenbanken vorhanden sind, ist ein Ansatz mithilfe von ML sehr hilfreich. Bis Dato gibt es noch sehr wenige Studien darüber Cross-Selling Vorhersagen für Versicherungsunternehmen mittels ML zu erzielen. Der Unterschied von Versicherungsunternehmen (VU) zu anderen Unternehmen mit erfolgreichen Cross-Selling Algorithmen besteht darin, dass VU eine sehr überschaubare Palette an Produkten anbieten. Im Vergleich dazu haben Online-Händler oft tausende Produkte zur Auswahl. Die Wahl des richtigen Algorithmus kann somit stark von anderen Fallstudien zu dem selben Thema abweichen.

Ozdemir, Y.E. & Bayrakli, S. [5] beschreiben einen ML Ansatz unter der Benutzung von *Decision-Trees*. Das Ziel ihrer Vorgehensweise ist es die Wahrscheinlichkeit eines Versicherungsnehmers zu berechnen, ob er als Nächstes das ausgewählte Nicht-Lebens-Produkt R1

kauft oder nicht. Das Eingangsdatenset wurde in vier verschiedene Kategorien unterteilt: Demographische Dimension, Finanzdaten, Kundenkontakte, Produkt Informationen und zusätzliche Parameter resultierend aus anderen analytischen Modellen. Die Parameter mit den meisten Einfluss waren das Werte- und Verhaltenssegment und die Rücklaufquoten der Kunden. Die Versicherungsnehmer mit den höchsten Kaufwahrscheinlichkeiten wurden anhand des präferierten Kanals kontaktiert und es wurde eine Steigerung von 25% für das Cross-Selling beobachtet.

Als Vorgehensweise, um den Kauf von zusätzlichen Kranken- und Lebensversicherungen vorauszusagen, wählen Sidorowicz, T. et al. [6] einen Ansatz mit *Positive Unlabelled Learning*, da nur 0,5% der Kunden beide Produkte gekauft haben. Aufgrund dessen ist ein Ansatz ohne die Daten als Cross-Sold zu deklarieren empfehlenswerter. Die Vorgehensweise baut auf Textähnlichkeiten aus Kunden-Berater-Interaktionsskripten auf. Die Ergebnisse zeigen, dass ein Bagging-Meta-Schätzer in Kombination mit einem neuronalen Netz, Random-Forest oder LightGBM zu einem verlässlichen Klassifizierer führt. Das wurde anschließend in einem Empfehlungssystem integriert, damit das Unternehmen die erkannten potenziellen Cross-Selling Kunden gezielter beraten kann.

Sekeroglu, A.G. [7] untersuchte einen ML Ansatz mit vorhergehendem *Feature Selection* als *preprocessing step* (vgl. 2.1). Das Ziel ist es vorherzusagen wie wahrscheinlich ein Versicherungsnehmer mit einer Krankenversicherung, ein KFZ-Versicherungsprodukt kauft. Die Studie untersuchte 7 verschiedene Algorithmen mit jeweils 4 verschiedenen *Feature Selection* Methoden und einer Technik zur Dimensionsreduktion. Durch die verschiedenen *preprocessing steps* haben die unterschiedlichen ML Algorithmen auch anders abgeschnitten und es konnte kein Algorithmus definitiv als bester ausgewählt werden. Aber die Zeit zum Trainieren der Algorithmen wurde drastisch reduziert, sodass eine solche *Feature Selection* Methode vor dem Benutzen eines ML Algorithmus empfehlenswert ist.

Weitere Studien sind Qazi, M. et al. [8] welche ein Bayesian Network, zum prognostizieren von Versicherungsprodukten für Neu- und Bestandskunden, vorschlagen. Oder auch Rokach, L. et al. [9] welche ein *Item-Item Collaborative Filtering* Ansatz, zur Vorhersage von Zusatzversicherungen, verfolgen.

## 2.3. Recommender Systems

Wie von oben zu entnehmen, wird im Allgemeinen zur Benutzung von sogenannten *Recommender Systems* (zu deutsch auch oft Empfehlungsdienst genannt), ein *Collaborative Filtering* (CF) Ansatz angewandt. Des Öfteren kommen auch *Content-Based Filtering* oder *Knowledge-Based Systems* zum Einsatz. Der Fokus in dieser Arbeit liegt auf dem CF Ansatz. Dabei werden verschiedene CF Methoden getestet und deren Ergebnisse miteinander verglichen.
CF kann man als die Vorgehensweise beschreiben, die eine automatisierte Prognose eines Kunden unter gewissen Bedingungen herleiten kann. Um so eine Vorhersage treffen zu können, wird eine sehr große zugrundeliegende Datenbank mit Informationen zum Kun-

den benötigt. Hier liegt genau die Schnittstelle zwischen *Data Mining* und *Collaborative Filtering*. Grundsätzlich gibt es verschiedene Arten von CF, zum einen den speicherbasierten Ansatz und zum anderen den modellbasierten. Seit längeren haben sich auch Hybrid Modelle durchgesetzt wie z.B. das von Google zur Nachrichten Empfehlung. Die neueste Methode, die sich in den letzten Jahren immer wieder bewährt hat und starke Aufmerksamkeit in den Medien erzielte, sind die *Deep-Learning* Techniken. Diese kommen immer wieder in den verschiedensten Szenarien auf und weisen meist eine sehr starke Lernrate auf. Je nach Aufgabenstellung ergeben sich unterschiedliche Probleme und Herausforderungen mit denen man konfrontiert ist. Dafür ist es meist hilfreich das Ziel des *Recommender Systems* genau zu formulieren und daraus mögliche Herausforderungen abzuwägen, um damit gezielt die richtige Methode auszuwählen.
Die in dieser Arbeit zu behandelnde Frage lässt sich so formulieren: Mit welcher Methode gelingt es am besten, einen Privatkunden der Allianz Elementar Vers. AG ein zusätzliches Risikosegment vorzuschlagen, um diesen damit ein, mit hoher Wahrscheinlichkeit, zusätzliches Versicherungsprodukt zu verkaufen?
Anstelle eines produktionsorientierten Ansatzes wird ein ausgewähltes Risikosegment des Versicherungsnehmers als Zielvariable genommen. Der Grund dafür ist, dass Produkte bei Versicherungen oft nur einer gewissen Produktgeneration zugeordnet sind, welche verschiedene Bestandsteile haben und oft auch andere Nebenrisiken abdecken, ein Risikosegment bleibt jedoch über Jahre hinweg dasselbe und eignet sich somit besser zum Projizieren in die Zukunft. Die genaue Definition zur Umsetzung der hier formulierten Fragestellung wird in Kapitel 4 behandelt.
Nun gilt es aus den oben kurz vorgestellten *Recommender Systems*, den für diese Frage besten Ansatz auszuwählen. Wie im Artikel von Su, X. & Khoshgoftaar, T.M. [10] beschrieben sind die Vorteile des speicherbasierten CF eine einfache Implementierung. Außerdem können neue Daten einfach hinzugefügt werden und die Skalierung mit korrelierten Produkten ist sehr gut. Dafür braucht es jedoch ein von Menschen abgegebenes Rating. Die Präzision der Vorhersage lässt bei spärlichen Datenquellen nach, die vorgestellte Methode lässt sich nicht auf hohe Datenmengen skalieren und können keine neuen Kunden und Produkte empfehlen. Das modellbasierte CF eignet sich für große und spärliche Daten besser. Es hat eine stärkere Prognosekraft und gibt weitgehend eine intuitive Begründung für die getroffenen Empfehlungen. Das klingt im ersten Moment sehr vielversprechend, aber auch diese Art des CF hat seine Nachteile, denn eine bessere Performance liegt meist im Ausgleich mit der Skalierbarkeit. Dadurch, dass diese Modelle besser für große Datenmengen sind liegt an ihrer Methodik, welche zugleich auch eine mathematische Komplexität mit sich zieht und dadurch die Implementierung um einiges schwieriger macht. *Deep-Learning* zeichnet sich auch durch eine hohe Skalierbarkeit aus, wird aber oft als Black-Box beschrieben und lässt sich nicht mehr durch eine rationale und nachvollziehbare Entscheidungskette ausdrücken. Trotzdem führt die hohe Genauigkeit zu oft überraschend guten Ergebnissen und ist somit für die Fragestellung geeignet. Des weiteren sind die modellbasierten CF auch von großem Interesse, da diese mit vielen und spärlichen Datenmengen umgehen können.

# 3. Mathematische Modelle und Methoden

In diesem Kapitel werden die für diese Diplomarbeit ausgewählten Modelle bzw. Algorithmen von Risikosegmenten näher erläutert und dessen mathematische Formulierungen dargelegt. Um die beste Methode für das Cross-Selling von Versicherungsprodukten auszuwählen, werden die folgenden Modellbasierten CF Klassifizierungs-Algorithmen miteinander verglichen:

- 3.1 Naive Bayes Klassifizierer
- 3.2 Logistische Regression
- 3.3 Linear Discriminant Analysis
- 3.4 Decision Trees
- 3.5 Random Forest
- 3.6 Gradient Boosting (CatBoost, LightGBM, XGBoost)
- 3 Deep-Neural-Networks

In diesem Abschnitt wird vorausgesetzt, dass ein historischer Datensatz $\Omega = (F, C)$ für jeden privaten VN $1 \leqslant i \leqslant I$ zur Verfügung steht. Das Risikosegment welches als Zielvariable festgelegt wurde, wird mit $C$ bezeichnet und kann die Werte $C_1 = 1$ oder $C_0 = 0$ annehmen (Risikosegment verkauft oder nicht verkauft). Die Elemente der Matrix F sind die zugehörigen $n$ verschiedenen Variablen zur Beschreibung des Kaufverhaltens des Kunden, diese können von numerischer sowie kategorischer Natur sein. Die Behandlung derverschiedenen Datenarten wird in Kapitel 4 näher beschrieben.

## 3.1. Naive Bayes Klassifikator

Der aus dem Satz von Bayes hergeleitete Klassifikator ist einer der effektivsten und effizientesten induktiven Lernalgorithmen für *Machine Learning* und *Data Mining*, insbesondere in Bezug auf seine Leistung bei Klassifizierungsaufgaben. Ein Grund dafür ist, dass er als optimaler Klassifikator angesehen wird. Er minimiert die durchschnittliche Fehlerwahrscheinlichkeit bei der Klassifizierung [11]. Dabei setzt dieser Klassifikator voraus, dass Informationen über Klassen in der Form von apriori Wahrscheinlichkeiten und Verteilungen bekannt sind. Die Bezeichnung „Naive “kommt daher, dass der Bayes-Klassifizierer die starke Annahme vertritt, dass die Klassen bzw. Features untereinander unabhängig sind, was sicherlich in der Praxis in den seltensten Fällen der Fall ist. In diesem Kapitel werden die Notation und die wichtigsten mathematischen Behauptungen beschrieben und erklärt

warum der Klassifizierer trotz der starken Annahme so gute Genauigkeit liefert.
Bevor ein Maß zur Bestimmung der Fehlerwahrscheinlichkeit eingeführt werden kann, bedarf es eines kurzen Exkurses zu dem Thema Satz von Bayes.
Sei $F_i$ der Zeilen-Vektor aus dem historischen Datensatz $F$ mit $n \in \mathbb{N}$ Elementen, wo $n$ die Anzahl der Features ist und $1 \leqslant i \leqslant I$ der i-te VN in den Grunddaten. Weiters sei $C_i$ die Hypothese, dass $F_i$ zu einer gewissen Klasse zugewiesen wird. Unter der Annahme, dass die apriori Wahrscheinlichkeiten $\mathbb{P}(C_i)$ und $\mathbb{P}(F_i)$ und die bedingte Wahrscheinlichkeit $\mathbb{P}(F_i|C_i)$ gegeben sind, besagt der Satz von Bayes:

$$\mathbb{P}(C_i|F_i) = \frac{\mathbb{P}(F_i|C_i)\mathbb{P}(C_i)}{\mathbb{P}(F_i)} \tag{3.1}$$

für Szenarien $F_i$ und $C_i$ mit $\mathbb{P}(C_i) \neq 0$ und $\mathbb{P}(F_i) \neq 0$. Dieses Resultat folgt direkt aus der Definition der bedingten Wahrscheinlichkeit. Redet man nun von einem einheitlichen Muster zu den gegebenen Klassen, dann lässt sich aus (3.1) und

$$\mathbb{P}(F_i) = \sum_k \mathbb{P}(F_i|C_i = C_k)\mathbb{P}(C_i = C_k)$$

eine Formel zur Berechnung der posteriori Wahrscheinlichkeit $\mathbb{P}(C_i|F)$ formulieren.

### 3.1.1. Minimum Fehler Klassifizierer

Wenn Muster in Kategorien eingeteilt werden, ist es hilfreich zu wissen wie stark der Fehler bei der Klassifizierung ist. Es ist wichtig, die Muster in ihrer Gesamtheit zu betrachten und die Wahrscheinlichkeiten der einzelnen Muster zu berücksichtigen. Für ein Testsatz $Y$ lässt sich die posteriori Wahrscheinlichkeit für die Klasse $C$ finden. Sei die Testmenge $Y \subseteq \Omega \backslash C$ der Klasse $C$ zugeordnet, dann lässt sich daraus schließen, dass die Fehlerwahrscheinlichkeit $1 - \mathbb{P}(C|Y)$ ist. Daraus folgt, dass zur Minimierung des Fehlers ein Testmuster der Klasse zugeordnet wird, für das die Wahrscheinlichkeit $\mathbb{P}(C|Y)$ maximal ist. Weiters lässt sich der Minimum Fehler Klassifizierer wie folgt definieren [12]:
Die erwartete Fehlerwahrscheinlichkeit ist für diskrete Werte gegeben durch

$$\mathbb{E}[\mathcal{F}] = \sum_{F_i}(1 - \mathbb{P}(C_i|F_i))\mathbb{P}(F_i)$$

### 3.1.2. Naive Bayes Wahrscheinlichkeitsmodell

Mit Hilfe des Satzes von Bayes (3.1) lässt sich das Modell zur Klassifizierung wie folgt ausformulieren:

$$\mathbb{P}(C|F_1, \ldots, F_n) = \frac{\mathbb{P}(F_1, \ldots, F_n|C)\mathbb{P}(C)}{\mathbb{P}(F_1, \ldots, F_n)} \tag{3.2}$$

Der Nenner ist nicht von den Klassen $C$ abhängig, kann für die Testmenge $Y = (F_1, \ldots, F_n)$ berechnet werden und bleibt darin immer konstant. Die Multiplikation im Zähler ist de fakto die gemeinsame Wahrscheinlichkeit von $(C, F_1, \ldots, F_n)$ und kann mit der Definition

der bedingten Wahrscheinlichkeit wie folgt umformuliert werden:

$$\begin{aligned}\mathbb{P}(C_i, F_{i,1}, \ldots, F_{i,n}) &= \mathbb{P}(C_i)\mathbb{P}(F_{i,1}, \ldots, F_{i,n}|C_i) \\ &= \mathbb{P}(C_i)\mathbb{P}(F_{i,1}|C_i)\mathbb{P}(F_{i,2}, \ldots, F_{i,n}|C_i, F_{i,1}) \\ &\vdots \\ &= \mathbb{P}(C_i)\mathbb{P}(F_{1,1}|C_i)\mathbb{P}(C_i)\mathbb{P}(F_{i,2}|C_i, F_{i,1}) \ldots \mathbb{P}(F_{i,n}|C_i, F_{i,1}, \ldots, F_{i,n-1})\end{aligned}$$

Laut Annahme sind die Features untereinander bedingt unabhängig, d.h. es gilt für alle $F_{i,j}$ mit $1 \leqslant j \leqslant n$:

$$\mathbb{P}(F_{i,l}|C_i, F_{i,m}) = \mathbb{P}(F_{i,l}|C_i), \qquad l \neq m$$

Damit kann die Klassifizierung durch das Naive Bayes Modell wie folgt ausgedrückt werden:

$$\mathbb{P}(C_i|F_{i,1}, \ldots, F_{i,n}) = \frac{1}{K}\mathbb{P}(C_i)\prod_{j=1}^{n}\mathbb{P}(F_{i,j}|C_i)$$

mit der Konstante $K = \mathbb{P}(F_{i,1}, \ldots, F_{i,n})$ als Skalierungsfaktor innerhalb einer gegebenen Testmenge. Ein Modell in dieser Form lässt sich viel besser handhaben als wie in (3.2), da nur die a priori Wahrscheinlichkeiten der Klassen $\mathbb{P}(C)$ und $\mathbb{P}(F_i|C)$ geschätzt werden müssen. Kann weiters die Wahrscheinlichkeit der einzelnen Features $\mathbb{P}(F_i)$ in $r$ verschiedenen Parameter beschrieben werden, dann müssen in Summe somit bei einem Modell mit 2 Klassen (gekauft oder nicht gekauft) $2nr + 1$ Parameter berechnet werden.

### 3.1.3. Naive Bayes Klassifikator

Der Naive Bayes Klassifikator kombiniert nun die Resultate aus den obigen Überlegungen und führt eine Entscheidungsregel zum Wahrscheinlichkeitsmodell hinzu. Eine gebräuchliche Regel besteht darin, die Hypothese auszuwählen, die die höchste Wahrscheinlichkeit aufweist, was als Maximum-a-Posteriori (MAP) bekannt ist. Der entsprechende Klassifikator ist die Funktion welche ein Klassen-Label $\hat{y} = C_k$ für beliebige $k$ wie folgt zuweist:

$$\hat{y} = \arg\max_{k \in \{0,1\}} \mathbb{P}(C_i = C_k)\prod_{l=1}^{n}\mathbb{P}(F_{i,l} = f_l|C_i = C_k)$$

Obwohl die sehr starke Annahme der bedingten Unabhängigkeit von den einzelnen Features oft sehr ungenau ist, ermöglicht die Trennung der Merkmalsverteilungen nach Klassen, dass jede Verteilung unabhängig als eindimensionale Verteilung geschätzt werden kann. Das verringert die dafür notwendige Rechenleistung enorm und hilft unter anderem besser mit einer hohen Datenmenge umzugehen.

## 3.2. Logistische Regression

Das statistische Verfahren der Regressionsanalyse spielt eine große Rolle bei der Untersuchung von Beziehungen zwischen mehreren unabhängigen Variablen. Ihr Ursprung liegt darin, komplexe Phänomene durch die Identifikation anderer einfacher messbaren Variablen

zu erklären. Dieses Verfahren ermöglicht es somit Muster, Trends und Zusammenhänge in *Data Mining* Schritten zu erkennen und auch mathematisch zu modellieren. Dies dient im Allgemeinen für die Berechnung von Prognosen und Entscheidungsfindungen ohne Einschränkung auf einen Themenbereich. Aus mathematischer Sicht ist die Regressionsanalyse ein Versuch eine, im einfachsten Fall, Gerade durch die von den zu untersuchenden Daten gebildete Punktwolke zu ziehen. Im Wesentlichen wird auf der Annahme gebaut, dass es eine Beziehung zwischen einer abhängigen Variable $C$ und mehreren unabhängigen Variablen $F_1, \ldots, F_n$ gibt. Infolgedessen kann für die lineare Regression, mit einem kleinen Fehlerterm $\epsilon$ folgende Form nieder geschrieben werden:

$$C = \beta_1 F_1 + \ldots + \beta_n F_n + \epsilon \tag{3.3}$$

Die Koeffizienten $\beta_1, \ldots, \beta_n$ werden als Regressionskoeffizienten bezeichnet und $\epsilon$ ist der Fehlerterm. Diese Koeffizienten werden so gewählt, dass der Abstand zu der Punktewolke möglichst minimal ist, eine gewöhnliche Methode zur Bestimmung des Abstandes ist die Methode der kleinsten Quadrate. Die lineare Regression hat jedoch keine Einschränkung für die Zielvariable $C$. Zur Bestimmung von Cross-Selling Kunden wird aber eine Wahrscheinlichkeit und damit eine Einschränkung auf das Einheitsintervall $[0, 1]$ benötigt. Anders gesagt ist die Zielvariable dafür $C_i \sim Ber\{p_i\}$. In diesem Fall kann (3.3) geschrieben werden als:

$$\begin{aligned} \mathbb{E}[C_i|F_{i,1}, \ldots, F_{i,n}] &= \beta_1 F_{i,1} + \ldots + \beta_n F_{i,n} + \epsilon \\ \mathbb{E}[C_i|F_i] &= \beta F_i \\ \mathbb{E}[C_i|F_i] &= \mathbb{P}[C_i = 1|F_i] = \beta F_i \end{aligned}$$

mit $\beta \in \{\beta_1, \ldots, \beta_n, \epsilon\}$. Die letzte Gleichung gilt dabei nur für Bernoulli Zufallsvariablen. Nun gilt es noch die Regressionsschätzung durch die lineare Funktion $\beta F$ durch eine Funktion zu ersetzen welche garantiert, dass das Ergebnis im Einheitsintervall bleibt und somit die Gleichung der bedingten Wahrscheinlichkeit gewährleistet. Die Standard Logistische Funktion, welche diese Überlegungen zu Papier bringt, erhält man aus der Inversen der sogenannten Logit-Funktion:

$$logit\{C_i = 1\} = \ln[Chancen\ so dass\ C_i = 1] = \ln\left[\frac{\mathbb{P}(C_i = 1|F_i)}{(1 - \mathbb{P}(C_i = 1|F_i)}\right]$$

Der Fehlerterm $\epsilon$ wird stets so gewählt, dass $p_i \notin \{0, 1\}$ und somit kann nach Funktion nach $p_i = \mathbb{P}(C_i|F_i)$ aufgelöst werden. Sei die Logit-Transformation im weiteren mit $x$ bezeichnet,

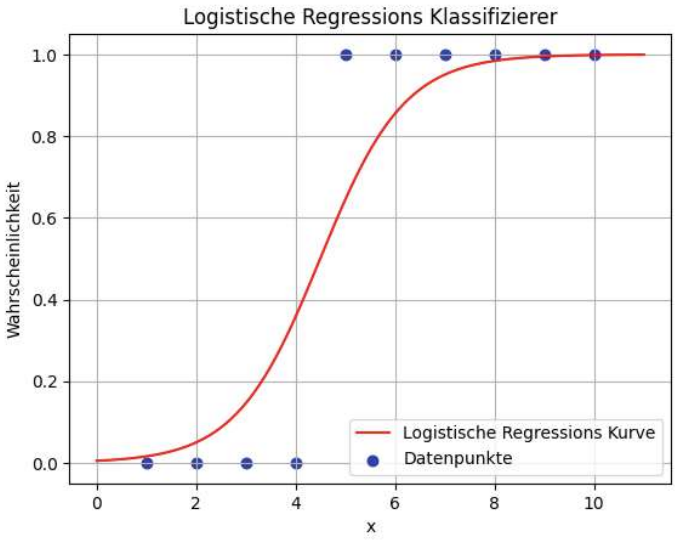


Abbildung 3.1.: Beispiel eines Logistischen Regressions Klassifizierer für eine kleine Menge an Datenpunkten

dann gilt:

$$\begin{aligned} x &= \ln[\frac{p_i}{1-p_i}] \\ -x &= \ln[\frac{1-p_i}{p_i}] \\ \exp^{-x} &= \frac{1-p_i}{p_i} \\ \exp^{-x} &= \frac{1}{p_i} - 1 \\ p_i &= \frac{1}{(1+\exp^{-x})} \end{aligned}$$

und diese Logistische Funktion bildet $\mathbb{R}$ auf $(0, 1)$ ab und garantiert somit das Wahrscheinlichkeitsmaß. In Summe lässt sich damit eine generelle logistische Regression formulieren:

$$C_i = \mathbb{E}[C_i|F_i] + \epsilon\mathbb{E}[C_i|F_i] = \mathbb{P}[C_i|F_i] = p_i = \frac{1}{1+\exp^{-\beta F_i}} \tag{3.4}$$

Das heißt zur Bestimmung einer binären Klassifizierung ist die logistische Regression geeignet und benötigt lediglich die Paramter $\beta, \epsilon$. Zur Berechnung dieser Paramter wird üblicherweise die Maximum-Likelihood Methode herangezogen. Für die Maximum-Likelihoodfunktion $f_i(C_i) = p_i^{C_i}(1-p_i^{1-C_i})$ und unter der Annahme von unabhängigen Messungen erhält man:

$$L(C, \beta) = \prod_{i=1}^{I} f_i(C_i) = \prod_{i=1}^{I} p_i^{C_i}(1-p_i^{1-C_i})$$

Wie oft bei dieser Methode, ist der Logarithmus der Likelihood-Funktion wesentlich leichter handzuhaben und ändert nichts an dem Maximum, da der Logarithmus monoton wachsend ist.

$$\ell(C, \beta) = \ln L(C, \beta) = \sum_{i=1}^{I} C_i \ln \frac{p_i}{1 - p_i} + \sum_{i=1}^{I} \ln (1 - p_i)$$

mit der Logit-Transformation $x = \ln \frac{p_i}{1-p_i}$ und der vorherigen Herleitung der Umkehrfunktion $1 - p_i = \frac{\exp -x_i}{1+\exp -x_i}$ erhält man

$$\ell(C, \beta) = \sum_{i=1}^{I} C_i x_i \beta - \sum_{i=1}^{I} \ln (1 + \exp^{x_i \beta})$$

Durch Differenzieren und Null setzen erhält man somit die gewünschten Parameter zur Berechnung der logistischen Regression zur Bestimmung einer Bernoulli Zielvariable.

## 3.3. Linear Discriminant Analysis

Andere Methoden als wie die logistische Regression, zur Bestimmung von binären Zielvariablen wären zum Beispiel Probit Modelle oder *Linear Discriminant Analysis* (LDA). Letzteres ist aufgrund der einfacheren Berechenbarkeit die Methode, die sich mehr durchgesetzt hat. Dieser Vorteil kommt nicht ohne Grund, denn die LDA hat weit aus strengere Annahmen als die logistische Regression oder das Probit Modell. Das Modell zur Berechnung der LDA nimmt eine gegebene Verteilung von F gegeben C an, auch wenn dies nur in den seltesten Fällen bekannt ist. Die Logistische Regression im Vergleich rechnet mit einer direkten Verteilung und braucht deshalb keine weiteren Annahmen bzgl. der Verteilung. LDA ist eine statistische Methode zur Klassifizierung von Datenpunkten in verschiedene *Features* (Merkmale). Das Ziel der LDA ist es, eine Trennlinie zu finden, die die *Features* in einem Merkmalsraum am besten unterscheidet. Die Grundidee besteht darin, die Daten in einen Raum mit einer kleineren Dimension zu transformieren und dabei die Klassen unterscheidbar zu halten. Anders gesagt werden die Features auf eine Hyperebene projiziert, deshalb wird diese Methode oft auch in der Vorverarbeitung der Daten zur Dimensionsreduktion verwendet.

### 3.3.1. Diskriminanten-Funktion

In Kapitel 3.1 wurde mit Hilfe des Satzes von Bayes ein Klassifizierer hergeleitet. Dafür wird die bedingte Verteilung $\mathbb{P}(F_i|C_i)$ benötigt, welche explizit geschätzt werden muss. Die LDA verfolgt hingegen den Ansatz eine Diskriminanten-Funktion zu definieren, die einem *Feature* $F$ eine eindeutige Klassifizierungsentscheidung zuordnet. Im Folgenden wird die Diskriminanten-Funktion $g(F_i) : \mathbb{R}^n \to [0, 1]$, welche einen Feature Vektor der Größe $n$ auf eine Klasse $C_i$ projiziert, betrachtet. Es wird gesagt, dass der Klassifizierer die Features der Klasse $C_i$ zuordnet wenn:

$$g_l(F_i) > g_m(F_i), \forall l \neq m$$

Das heißt der Klassifizierer, bestehend aus einer Familie von Diskriminanten-Funktionen, wählt jene Klasse, die der größten Diskriminante entspricht. Zum Beispiel würde bei einem Bayes-Klassifizierer die entsprechende Diskriminante für einen Minimum-Fehler-Klassifizierer so definiert werden:

$$g_i(F_i) = \mathbb{P}(C_i|F_i)$$

Die maximale Diskriminante entspricht damit der Maximum-a-Posteriori Wahrscheinlichkeit aus Kapitel 3.1.
Betrachtet man jetzt den Fall für eine Zweidimensionale Klasse (VN kauft oder nicht), dann ist es von Vorteil, die Differenz der zwei dazugehörigen Diskriminanten $g_1, g_2$ zu definieren:

$$g(F_i) = g_1(F_i) - g_2(F_i)$$

Damit ist die neue Diskriminante $g$ in der Lage die Klasse $C_1$ auszuwählen falls $g(F_i) > 0$ und $C_2$ falls $g(F_i) < 0$, der Fall $g(F_i) = 0$ wird gesondert behandelt und je nach Gebrauch der einen oder anderen Klasse zugeordnet, in diesem Kapitel bleibt der Fall undefiniert und es wird nicht näher darauf eingegangen. Da es sich hierbei immer um ein Maximierungs-Problem handelt, ist es klar, dass eine Diskriminante mit beliebigen monoton wachsenden Funktionen verknüpft werden kann.

### 3.3.2. Lineare Diskriminanten-Funktion

Die Familie der linearen Diskriminanten-Funktionen zu den Features $F_i \in \mathbb{R}^n$ wird im Allgemeinen beschrieben als:

$$g(F_i) = wF_i + w_0 = \sum_{l=1}^{n} w_l F_{i,l} + w_0 \tag{3.5}$$

mit Gewichten $w, w_0$. Die Gleichung (3.5) beschreibt eine Hyperebene mit Richtung $w$ und senkrechter Richtung $\frac{|w_0|}{|w|}$. Es gilt bei der LDA also die Gewichte der Diskriminanten-Funktion für das gegebene Problem zu finden. Für den Fall, dass die Diskriminante einen Minimum Fehler Klassifizierer beschreibt, folgt zusammen mit voriger Beobachtung daraus:

$$\begin{aligned} g(F_i) &= \mathbb{P}(C_i = C_0|F_i) - \mathbb{P}(C_i = C_1|F_i) \\ g(F_i) &= \ln \frac{\mathbb{P}(F_i|C_0)}{\mathbb{P}(F_i|C_1)} + \ln \frac{\mathbb{P}(C_0)}{\mathbb{P}(C_1)} \end{aligned}$$

Für den Multidimensionalen Fall, also mit $k$ verschiedenen Klassen, ergibt sich mit dem Satz von Bayes für die einzelnen Diskriminanten und $1 \leqslant l \leqslant k$:

$$\begin{aligned} g_l(F_i) &= \mathbb{P}(C_i = C_l|F_i) \\ &= \frac{\mathbb{P}(F_i|C_i = C_l)\mathbb{P}(C_i = C_l)}{\sum_{j=1}^{k} \mathbb{P}(F_i|C_i = C_j)\mathbb{P}(C_i = C_j)} \end{aligned}$$

### 3.3.3. Diskriminanten-Funktion bei Normalverteilung

Sei im Folgenden wieder der Minimum Fehler Klassifizierer für die Diskriminante, also

$$g_l(F_i) = \ln \mathbb{P}(F_i|C_i) + \ln \mathbb{P}(C_i)$$

und die Verteilung der $F_i$ gegeben $C_i$ sei Normalverteilt mit

$$\mathbb{P}(F_i|C_i) \sim N(\mu_l, \Sigma_l)$$

und wir haben eine Dichte für die Features mit

$$p(F_i) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_l|^{\frac{1}{2}}} \exp -\frac{1}{2}(F-\mu_l)^t\Sigma_l^{-1}(F-\mu_i)$$

Daraus folgt für die Diskriminante

$$g_l(F_i) = -\frac{1}{2}(F_i-\mu_l)^t\Sigma_l^{-1}(F_i-\mu_l) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_l| + \ln \mathbb{P}(C_i)$$

Der Term $(F_i-\mu_l)^t\Sigma_i^{-1}(F_i-\mu_l)$ entspricht der quadratischen Mahalanobis-Distanz zwischen den Features $F$ aus dem Datenset und dem Mittelwert $\mu_l$. Die Mahalanobis-Distanz ist eine Mehrdimensionale Verallgemeinerung der Euklidischen-Distanz unter Berücksichtigung der Korrelation zwischen den verschiedenen Merkmalen. Die Zuordnung einer Klasse erfolgt somit basierend auf der Nähe des Mittelwerts dieser Klasse in Bezug auf die Mahalanobis-Distanz, wobei auch die A-Priori-Wahrscheinlichkeiten der Klassen miteinbezogen werden. Nun gilt es für die Funktionen $g_l$ die Gewichte $w_l$ und den Bias $w_0$ zu bestimmen. In der Praxis ist es oft hilfreich Annahmen für die Kovarianz-Matrix zu stellen, um damit eine schnellere Berechnung zu ermöglichen. Annahme 1 sei, dass alle Features untereinander unabhängig sind und jedes Feature dieselbe Varianz $\sigma^2$ hat. Annahme 2 sei, dass nur noch die Kovarianz-Matrix für jedes Feature gleich ist und Annahme 3 ist der allgemeine Fall mit beliebigen Kovarianz-Matrizen. Im Folgenden werden die drei Annahmen in drei Fälle unterteilt und die Gewichte/Bias für jeden hergeleitet.

Fall 1: *Linear Discriminant Analysis* (Abbildung 3.2 zeigt ein Beispiel dieses Klassifizierers)
Dies ist der einfachste Fall, da am meisten Annahmen hierfür getroffen werden. Dadurch, dass alle *Features* mit der selben Varianz angenommen werden, ergibt sich für $|\Sigma_i| = \sigma^{2d}$ und für die Inverse $\Sigma_l^{-1} = \frac{1}{\sigma^2}I$. Damit sind die Terme $|\Sigma_l|$ und $\frac{d}{2}\ln 2\pi$ unabhängig von $l$ und können vernachlässigt werden, damit ergibt sich

$$\begin{aligned} g_l(F_i) &= -\frac{||F_i-\mu_l||^2}{2\sigma^2} + \ln \mathbb{P}(C_i) \\ &= -\frac{1}{2\sigma^2}({F_i}^t F_i - 2\mu_l^t F_i - \mu_l^t\mu_l) + \ln \mathbb{P}(C_i) \end{aligned}$$

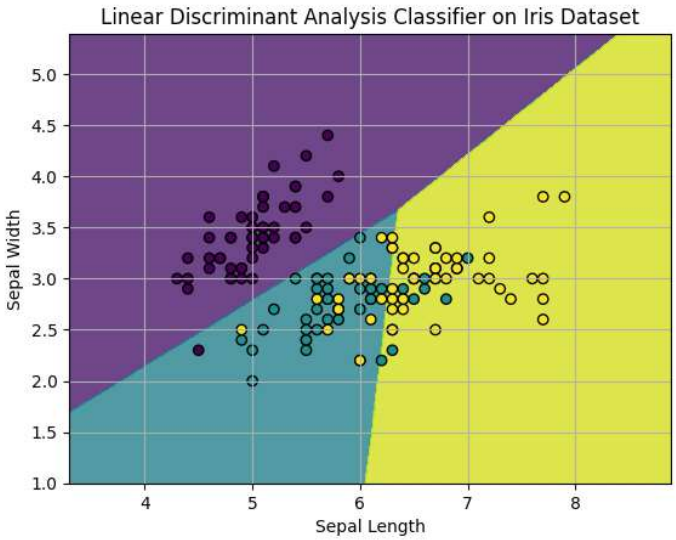


Abbildung 3.2.: Beispiel eines Linear Discriminant Analysis Klassifizierer, trainiert auf dem Iris Beispieldatensatz mit den Merkmalen 'Sepal Width' und 'Sepal Length'

Da laut Annahme $F^tF$ unabhängig von $i$ ist, kann dieser Term vernachlässigt werden. Damit lässt sich jetzt die Diskriminante in geschlossener linearer Form darstellen:

$$g_l(F_i) = wF_i + w_0$$
$$w_i = \frac{1}{\sigma^2}\mu_l$$
$$w_{i0} = -\frac{1}{2\sigma^2}(\mu_l^t\mu_l) + \ln \mathbb{P}(C_i)$$

Der Punkt an dem zwischen zwei verschiedenen Klassen unterschieden wird, ist gegeben durch $g_l(F_i) = g_m(F_i)$. Das wird oft auch als Hyperebene oder Entscheidungsfläche (*Decision Surface*) bezeichnet und kann mit einfachen Äquivalenzumformungen umgeschrieben werden zu

$$w(F_i - x_0) = 0$$
$$w = \mu_l - \mu_m$$
$$x_0 = \sigma^2(w_{l,0} - w_{m,0}) =$$
$$= \frac{1}{2}(\mu_l - \mu_m)\frac{\sigma^2}{||\mu_l - \mu_m||^2}\ln\frac{\mathbb{P}(C_l)}{\mathbb{P}(C_m)}(\mu_l - \mu_m)$$

Fall 2: *Quadratic Discriminant Analysis* (3.3 zeigt ein Beispiel dieses Klassifizierers)
Nun sei die einzige Annahme nur mehr, dass die Kovarianz-Matrizen gleich für jede Klasse ist, also $\Sigma_l = \Sigma$. Damit ist der Teil der Disrkiminante mit $F\Sigma^{-1}F$ nicht mehr von $l$ abhängig

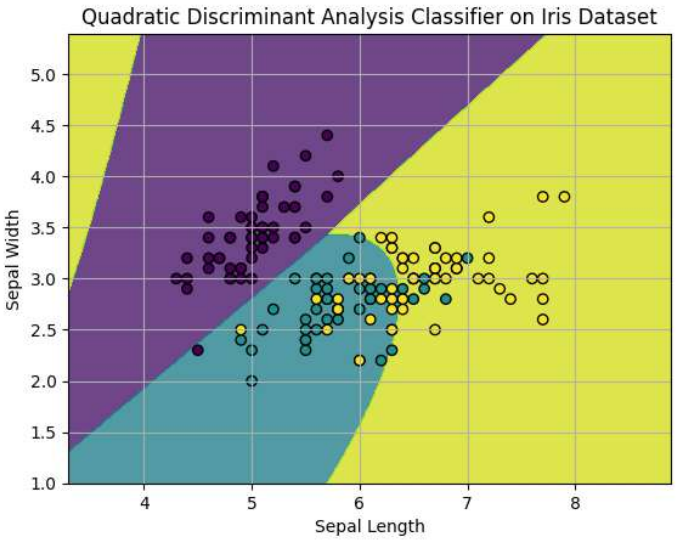


Abbildung 3.3.: Beispiel eines Quadratic Discriminant Analysis Klassifizierer, trainiert auf dem Iris Beispieldatensatz mit den Merkmalen 'Sepal Width' und 'Sepal Length'

und kann wieder vernachlässigt werden. Daraus lässt sich wieder eine geschlossene Form für die Gewichte und den Bias der linearen Diskriminante angeben:

$$g_l(F_i) = w_l F_i + w_{l,0}$$
$$w_l = \Sigma^{-1}\mu_l$$
$$w_{l,0} = -\frac{1}{2}\mu_l^t \Sigma^{-1}\mu_l + \ln \mathbb{P}(C_i)$$

Die Entscheidungsfläche $g_l(F) = g_m(F)$ ist somit

$$w(F_i - x_0) = 0$$
$$w = \Sigma^{-1}(\mu_l - \mu_m)$$
$$x_0 = \frac{1}{2}(\mu_l - \mu_m) - \frac{\ln \mathbb{P}(C_i)/\mathbb{P}(C_j)}{(\mu_l - \mu_m)^t \Sigma^{-1}(\mu_l - \mu_m)}(\mu_l - \mu_m)$$

Fall 3:
Der letzte und allgemeine Fall mit einer multivariaten Normalverteilung mit verschiedenen Kovarianz-Matrizen für jedes *Feature* führt dazu, dass am wenigsten der Terme vernachlässigt werden (nur der Term $\frac{d}{2}\ln 2\pi$ ist unabhängig von $i$). Dies führt unumgänglich dazu, dass die Berechnung der Diskriminanten und der Entscheidungsfläche sehr schnell sehr aufwändig wird. Damit ist dieser Fall für große Datenmengen oft unbrauchbar und wird in dieser Diplomarbeit nicht weiter verwendet. Der Vollständigkeit wird an dieser

Stelle trotzdem die geschlossene Form der linearen Diskriminante angeführt

$$g_l(F_i) = F_i^t W_l F_i + w_l F + w_{l,0}$$
$$W_l = -\frac{1}{2}\Sigma_i^{-1}$$
$$w_l = \Sigma_l^{-1}\mu_l$$
$$w_{l,0} = -\frac{1}{2}\mu_l^t\Sigma_l^{-1}\mu_l - \frac{1}{2}\ln|\Sigma_l| + \ln\mathbb{P}(C_i)$$

Aus dem Fall 1 sieht man, dass die LDA zu einer linearen Entscheidungsfläche führt und die QDA im Fall 2, welche keine identischen Kovarianzen annimmt, führt zu einer quadratischen Entscheidungsfläche. *Discriminant Analysis* Methoden sind also eine Art der Projizierung von einem n-dimensionalen Merkmal-Vektor auf eine Dimension $g_l$. Klarerweise wird durch eine solche Projektion auf eine viel niedrigere Dimension einiges an Information verloren gehen. Auch Klassen, welche im originalen Merkmalsraum linear separabel sind, müssen im Allgemeinen nicht mehr auf dem projizierten Vektor sein.

## 3.4. Decision Trees

Ein Entscheidungsbaum, oder auch *Decision Tree* (DT), ist eine *Supervised Learning* Methode zur Klassifizierung oder Regression. Er modelliert Entscheidungen in Form eines Baumes und wird durch Training mit gelabelten Daten erstellt, ohne dabei auf festgelegte Parameter oder Annahmen von Verteilungen zu beruhen. Das Ziel ist es, anhand der Datenmerkmale einfache Entscheidungsregeln zu erlernen, um daraus dann den Wert einer Zielvariable (Kauf oder nicht gekauft) vorherzusagen. Ein solcher Decision Tree kann also als eine stückweise Annäherungsmethode betrachtet werden, um Daten zu modellieren. Durch die Simplizität und ohne größere Annahmen weisen *Decision Trees* zahlreiche Vorteile auf. Sie können visuell dargestellt werden und sind somit leicht verständlich. Im Vergleich zu anderen Methoden erfordern sie weniger an Datenvorarbeit. Bestimmte *Decision Tree Algorithmen* können auch gut mit fehlenden Daten und Multi-Output-Problemen umgehen. Im Gegensatz dazu können so genannte Black-Box Modelle wie künstliche neuronale Netze schwieriger zu interpretieren sein. Die DT's lassen sich mithilfe statistischer Tests validieren, um die Zuverlässigkeit zu überprüfen, und zeigen oft gute Leistung. Dadurch zählen sie zu den beliebtesten Datenstrukturen bei Klassifizierungsproblemen[12]. Trotz der genannten Vorteile weisen *Decision Trees* auch einige Nachteile auf. Sie haben die Tendenz übermäßig komplexe Strukturen zu erzeugen. Sie werden dadurch schnell ineffizient und führen zu Overfitting.

### 3.4.1. Aufbau von Decision Trees

Ein *Decision Tree* wird aus einem Test-Datensatz aufgebaut. Dieser besteht aus einer Sammlung von Merkmalen und bereits zugeordneten Zielvariablen, was auch als labelled Dataset bekannt ist. Der Baum wird genau so gestaltet, um diesem Test-Datensatz bzw. Trainingset zu entsprechen und seine Zuordnungen und Muster nachzuahmen. Der daraus

abgeleitete *Decision Tree* ermöglicht die kompakte Darstellung vieler Fälle durch ein Flussdiagramm und dient als Beispiel für induktives Lernen. Um diesen Baum zu konstruieren, werden an jedem Knoten ein oder mehrere Merkmale ausgewählt um die Entscheidungsprozesse zu steuern. Die wichtigste Merkmalsentscheidung wird an den Anfang des Baumes gesetzt, sukzessive folgen weitere Entscheidungen mit einer geringeren Unterscheidungskraft. An jedem Knoten erfolgt eine Aufteilung des Datensatzes, wodurch wieder neue Verzweigungen entstehen. Eine gute Merkmalsentscheidung teilt die Datensätze in Untermengen auf, welche komplett positiv oder negativ sind. Im Gegenzug dazu ist eine Entscheidung schlecht, wenn es ein ähnliches Verhältnis von positiven und negativen Datensätzen wie das Original beibehält. Falls nun das angestrebte Ergebnis eine Klassifizierung mit den Werten $1, 2, \ldots, K$ ist, dann sind die einzigen Änderungen des *Decision Trees* die Kriterien für die Aufteilung der Knoten und das *Pruning* des Baumes. Die Aufteilung der Knoten muss dadurch auf Unreinheit gemessen werden. Dafür gibt es verschiedene *Impurity Measures*.

Auf einer Menge $(x_i, y_i)$ für $i = 1, \ldots, n$ sei die Anzahl der gemessenen Beobachtungen für die Klasse $k$ gegeben durch $I(y_i = k)$, damit ist der Anteil der Beobachtungen der Klasse $k$ im Knoten $m$ mit $N_m$ Beobachtungen in einer Region $R_m$ definiert durch:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{3.6}$$

Die Mehrheitsklasse im Knoten $m$ kann an dieser Stelle durch $k(m) = \arg\max_k \hat{p}_{mk}$ festgelegt werden. Sei weiters nun $T \subset T_0$ ein *Decision Tree* nach dem *Pruning* des größten Baumes $T_0$ mit allen möglichen Entscheidungen, dann sind die verschiedenen *Impurity Measures* für den getrimmten Baum:

$$\begin{aligned} Q_m^1(T) &= 1 - \hat{p}_{mk(m)} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i \neq k(m)) \\ Q_m^2(T) &= \sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \\ Q_m^3(T) &= -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk} \end{aligned}$$

$Q^1$ wird als Klassifizierungs-Fehler bezeichnet, $Q^2$ als Gini-Index und $Q^3$ als Log-Loss oder Cross-Entropy. Bei einem binären Klassifizierungsproblem ergeben diese drei Maße $1 - \max(p, 1-p), 2p(1-p)$ und $-p \log p - (1-p) \log(1-p)$ respektive, unter der Annahme, dass $p$ der Anteil in der zweiten Klasse ist. Während der Klassifizierungs-Fehler hier bei $p = 0,5$ auf einer Spitze durchläuft, bleiben der Gini-Index und die Cross-Entropy beide glatt. Im Allgemeinen sind die letzteren Maße differenzierbar und somit bei numerischen Optimierungen meist von Vorteil. Weiters sind die beiden auch sensibler bei Veränderungen in den Knoten-Wahrscheinlichkeiten. [13] Durch die Veränderung der *Impurity* $\Delta i(N) = i(N) - \sum_j (P_j i(N_j)$ für verschiedene Entscheidungen $j$ am Knoten $n$, kann somit ein *Informations-Gewinn* bzw. ein *Gain* gemessen werden. Jenes Attribut, das den *Gain* maximiert, gilt es als Nächstes zu wählen. Der erste veröffentlichte Klassifizierungs-Algorithmus für *Decision Trees* ist der THAID Algorithmus[14] von Messenger, R. & Mandell, L.. Dieser kann in einem kurzen Pseudocode, wie folgt, beschrieben werden:

| Alter | Geschlecht | Gehalt | Lieblingsfarbe | Kauft Produkt |
|---|---|---|---|---|
| 25 | Mann | 50.000€ | Blau | ja |
| 35 | Frau | 60.000€ | Rot | nein |
| 45 | Mann | 70.000€ | Grün | nein |
| 55 | Frau | 80.000€ | Blau | ja |
| 65 | Mann | 90.000€ | Rot | ja |
| 75 | Frau | 100.000€ | Grün | ja |
| 85 | Mann | 110.000€ | Blau | nein |
| 95 | Frau | 120.000€ | Rot | ja |

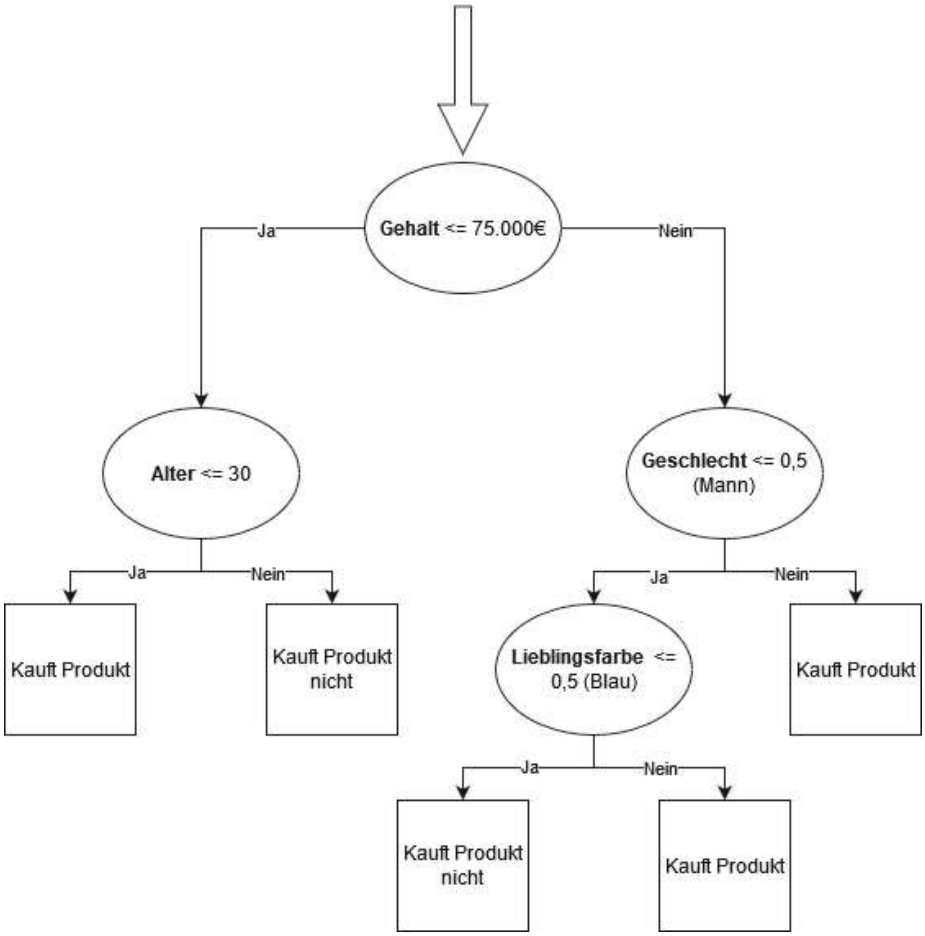


Abbildung 3.4.: Beispiel eines Decision Tree

**Algorithm 3** THAID Algorithmus

**Require:** Trainingsmenge $Y \subseteq \Omega = (F, C)$
**Ensure:** THAID Decision Tree
1: Start an dem Wurzelknoten
2: Für jedes $F$, finde die Menge $S$ welche die Summe der Knoten Impurities in den beiden folgenden Knoten minimiert und nehme das Splitting $\{F^* \in S^*\}$ welches das Minimum in $F$ und $S$ gibt.
3: **if** Stopp-Kriterium: $\Delta Impurity < \epsilon$ **then**
4: **return** Result
5: **else**
6: Wiederhole den Split in 2
7: **end if**

Weitere bekannte Algorithmen sind der C4.5 und CART, die auch auf diesem Pseudo-Algorithmus aufbauen. C4.5 benutzt die Cross-Entropy als Impurity Measure und der CART benutzt den Gini Index.

### 3.4.2. Splitting

Im Bereich der *Decision Trees* wird ein Entscheidungsergebnis an einem beliebigen Knoten als Splitting (bzw. Teilung) bezeichnet, wenn die Trainingsdaten in kleinere Sätze aufgeteilt werden. Der erste *Split* teilt somit den gesamten Datensatz auf und nachfolgende *Splits* setzten diese Unterteilung fort. Je nach den Bedingungen der Teilungsregeln werden die Daten auf unterschiedliche Weisen geteilt:

- *Axis-parallel split*: Eine Teilung durch einen Wert eines einzigen Merkmals

$$x_j > a_0$$

- *Oblique split*: Eine Teilung von mehreren Werte und Attributen durch eine linear Kombination. Bei einer binären Zielvariable würde ein Oblique Split durch die Lösung folgender Gleichung gegeben sein:

$$af_1 + bf_2 + c > 0$$

- *Multivariate Split*: Eine Teilung von mehreren Werten und Attributen durch eine lineare oder nicht-lineare Kombination von Merkmalen.

Die verschiedenen *Splits* werden solange nacheinander ausgeführt bis es zu einem vorgegebenen Stopp kommt. Eine häufig gewählte Methode zum Stoppen ist der Cross-Validation Test, hierfür wird im Vorhinein ein Teil des Trainingsmenge entfernt, um das Ergebnis des Decision Trees bzw. des Splitting zu validieren. Das kann auch mehrfach mit verschiedenen Untermengen der Testdaten, angewandt und abschließend ein Mittelwert der Performance gebildet werden. Eine weitere Methode zum Stoppen des *Splitting* ist, die Betrachtung der Veränderung der *Impurity*, d.h. wenn $\max_s i(s) \leqslant \beta$ oder anders gesagt, wenn das Impurity-Maß unter einem festgelegten Wert $\beta \in \mathbb{R}$ fällt, dann soll das *Splitting* gestoppt werden.

### 3.4.3. Overfitting und Pruning

Großen Datenmengen mit vielen verschiedenen Attributen folgen oft eine Vielzahl möglicher Hypothesen und es besteht die Gefahr, dass der induzierte *Decision Tree* zu stark wächst und somit zu spezifisch wird, dieses Phänomen wird als *Overfitting* bezeichnet. Eine Lösung für dieses Problem besteht in der Anwendung von *Pruning*, einer Methode, um die Entscheidungsbäume zu verkleinern. Dadurch soll verhindert werden, dass der Baum rekursiv anhand von Merkmalen aufgeteilt wird, die nicht eindeutig relevant sind. Die Algorithmen wie C4.5 und CART erzeugen als erstes einen riesigen *Decision Tree* und verfolgen erst am Ende die Anwendung von *Pruning*. CART benutzt z.B. eine *Cross-Validation* ähnlich wie beim Stopptest von der *Splitting* Methode. Eine andere Methode für das *Pruning* ist das Finden von unwichtigen Attributen. Auch in dieser Methode wird der Baum als allererstes komplett aufgebaut und erst im Nachhinein abgeschnitten. Angenommen, dass nach einem *Split* unter eines gewissen Attributes, die Menge der verschiedenen Klassen ungefähr das selbe Verhältnis hat, dann kann man davon ausgehen, dass dadurch nur eine geringe Menge an Information gewonnen werden kann. Im Allgemeinen kann man sagen, dass ein *Gain* von 0 auf ein irrelevantes Attribut hinweisen kann. Eine weitere Methode ist das sogenannte Minimal *Cost-Complexity Pruning*. Hier geht es darum, einen Teil-Baum von einen gegeben Baum $T$ zu finden, welcher das *Cost-Complexity* Maß

$$Q_\alpha(T) = Q(T) + \alpha|\tilde{T}|$$

minimiert, $|\tilde{T}|$ ist die Anzahl der End-Knoten und $Q(T)$ ist eines der *Impurity* Maße. Auf einzelnen Knoten ist dieses Maß dann $Q_\alpha(t) = Q(t) + \alpha$, wo $t$ der Wurzelknoten vom Teilbaum $t_t$ ist. Der effektive *Complexity Parameter* $\alpha \geqslant 0$ eines Knotens ist der Wert wo $Q_\alpha(T_t) = Q_\alpha(t)$ oder $\alpha_{eff} = \frac{Q(t)-Q(T_t)}{|T|-1}$ ist. Nun können sukzessive alle Knoten mit einem effektiven *Complexity Parameter* unter dem gewünschten Wert entfernt werden.

## 3.5. Random Forests

*Decision Trees* haben, wie im vorigen Kapitel erklärt, eine Schwäche gegenüber der Klassifizierung neuer Datensätze, oder anders gesagt, sie tendieren sehr schnell zu einem *Overfitting*. Auch durch die *Pruning-Methoden* wird dieser Schwäche zwar entgegengesetzt aber dennoch haben DT keinen systematischen Ansatz dieses Risiko zu minimieren. Der erste erfolgreiche Versuch mithilfe von stochastischer Modellierung eine neue Methode für Baumstrukturen zu konstruieren gelang Tim Kam Ho [15]. Ho's Ansatz beruht darauf, nicht nur einen Tree zu erzeugen, sondern mehrere aus zufällig ausgewählten Teilmengen des Merkmalraumes. Leo Breiman [16] und Adele Cuttler [17] erweiterten den Ansatz von Ho und führten die von Breiman beschriebene *Bagging-Methode* zur *Random Subspace Methode* hinzu. Dadurch sollen viele unkorrelierte *Trees* erzeugt werden und dadurch die Varianz der Klassifizierung sinken. Die Aussage mit den meisten Stimmen klassifiziert das zu prognostizierende Ergebnis. *Random Forests* behalten dabei die einfache Nachvollziehbarkeit der DT und sind einfach zu trainieren und verbessern.

### 3.5.1. Bootstrap and Bagging

Das Konzept von *Bootstrap* ist aus den Anfangs Datenbestand zufällige Zeilen herauszuziehen, um damit einen neuen *Bootstrap*-Datensatz mit gleicher Dimension zu erzeugen, dabei darf jede Zeile auch öfters ausgewählt werden. Dieser Vorgang wird anschließend $B$-mal wiederholt und somit entstehen $B$ verschiedene Datensätze. Um das Modell durch diese Datensätze zu verbessern, muss ein Vorhersagefehler geschätzt werden. Eine Möglichkeit wäre, das Modell an eine Reihe von *Bootstrap*-Stichproben anzupassen und zu prüfen, wie gut es den Anfangs Datenbestand vorhersagt. Sei $\hat{f}_i^{*b}$ die Vorhersage für den $i$-ten VN, dann wäre der Fehler über alle *Bootstrap*-Bestände:

$$\widehat{ERR}_{boot} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{I} L(C_i, \hat{f}_i^{*b})$$

Da es bei diesem Vorgang jedoch Gemeinsamkeiten zwischen Test- und Trainingssätzen gibt, kann es zu unrealistisch guten Vorhersagen führen. Die Lösung dafür ist eine Validierung mit nicht überlappenden Daten für Test- und Trainingssätzen durchzuführen.
Die *Bagging*-Methode fügt dem *Bootstrap*-Verfahren noch ein Aggregationsverfahren hinzu, es wird somit eine Vorhersage aus dem Mittelwert von der Sammlung von *Bootstrap*-Beständen getroffen. Die Grundidee kommt daher, dass der *Bootstrap*-Mittelwert sich dem *Posterior-Bayes-Mittelwert* annähert [13]. Der Schätzer für die *Bagging*-Vorhersage für den $i$-ten VN ist also:

$$\hat{f}_{ibag}(F_i) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_i^{*b}$$

Aufgrund des Gesetzes der Großen Zahlen leiden *Random Forests* bei großen $B$ nicht unter Overfitting, sondern nähern sich einem Limit des *Generalization-Errors*[16]. Angenommen ein *Tree* erzeugt einen Klassifikator $\hat{G}(F_i)$ für eine Klassifizierungs-Aufgabe mit $K$ verschiedenen Klassen. Weiters sei $\hat{f}(F_i)$ eine Indikatorfunktion zur Identifizierung der Klasse, so dass $\hat{G}(F_i) = \arg\max \hat{f}(F_i)$. Dann ist die Schätzung des *Baggings* ein Vektor mit:

$$\hat{f}_{bag}(F_i) = [p1(F_i), p2(F_i), \ldots, pK(F_i)]$$

wobei $pk(F)$ der Anteil der Trees ist, die bei $F$ die Klasse $k$ vorhersagen. Der *Bagging* Klassifier wählt abschließend die Klasse mit den meisten Stimmen aus den $B$ verschiedenen Trees.

$$\hat{G}_{bag}(F_i) = \arg\max_k \hat{f}_{bag}(F_i)$$

Oft wird bei Klassifizierungs-Vorhersagen nicht nur eine Klassifizierung, sondern eine Wahrscheinlichkeitsschätzung der Klassen benötigt. Dafür wäre eine alternative *Bagging*-Strategie, die Funktionen $\hat{f}(F_i)$ zu mitteln, die bereits die Klassenwahrscheinlichkeiten schätzen. Insbesondere für kleine $B$ werden dadurch bessere Schätzungen der Klassen und auch geringere Varianzen der *Bagging*-Klassifikatoren erzielt.

### 3.5.2. Definition

Durch das Anwenden des oben beschriebenen *Bagging*, können stark rauschende und *unbiased* Modelle durch den Mittelwert eine geringere Varianz erzielen. *Decision Trees* sind

durch ihr starkes Rauschen ein idealer Kandidat für den *Bagging*-Ansatz. Werden etwa $B$ solcher *Trees* erzeugt und seien diese unabhängig gleichverteilt mit Varianz $\sigma^2$ für einen *Tree*, dann gilt für die Varianz nach *Bagging* (Beweis: A.0.1):

$$\frac{1}{B}\sigma^2$$

und falls diese nicht unabhängig aber gleichverteilt sind, dann gilt (Beweis: A.0.2):

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Da nun bei nicht unabhängigen Trees die Varianz sich für große $B$ an $\rho\sigma^2$ annähert, hat der Aggregations-Effekt nur einen begrenzten Vorteil zur Varianz-Minimierung. *Random Forests* versuchen zusätzlich zum Aggregieren, die Korrelation der Bäume untereinander zu verringern, um die Varianz noch mehr zu reduzieren. Dies wird mithilfe von *Bootstrap Splits* erzielt. Die Vorgehensweise zur Erstellung eines *Random Forest* für Klassifizierung kann somit wie folgt beschrieben werden:

**Algorithm 6** Random Forest für Klassifizierung

**Require:** Trainingsmenge $Y \subseteq \Omega = (F, C)$
**Ensure:** Random Forest Klassifizierer
1: **while** $1 \leqslant b \leqslant B$ **do**
2: Erzeuge ein Bootstrap-Dataset mit Mehrfachauswahl und gleicher Dimension wie die Grunddaten
3: **while** Stopp-Kriterium: $Knotengrad > \epsilon$ **do**
4: Erzeuge einen Random Forest Tree $T_b$ aus dem Bootstrap-Dataset indem zu erst eine zufällige Teilmenge der Features und wähle das Feature mit dem besten Split gewählt wird.
5: **end while**
6: **end while**
7: Validiere den Tree Ensemble $\{T_b\}_1^B$ mit den Trainingsdaten und die Klasse mit den meisten Stimmen ist der Random-Forest Klassifizierer:

$$\hat{C}_{RF}^B(F_i) = \arg\max_b \hat{C}_b(F_i)$$

### 3.5.3. Out-of-Bag

Beim ersten Schritt zum Erzeugen eines *Random Forest* wird wie oben beschrieben ein *Bootstrap-Dataset* erstellt. Da beim Bootstrapping auch einzelne Zeileinträge öfters vorkommen können, stellt sich die Frage was mit den übrig gebliebenen Daten passiert. Es hat sich bewährt, die sogenannten *Out-of-Bag* (OOB) Daten zum Validieren und Verbessern des *Random Forest* zu verwenden. Durch die OOB kann ein OOB-Fehler für den RF geschätzt werden. Damit kann in weiterer Folge der RF besser konstruiert werden. Da durch den OOB Datensatz nun die gesamte Grundmenge an Informationen benutzt wird, ist klar, dass durch die Verwendung des OOB-Fehler eine bessere Performance als wie bei sonst

üblichen Cross-Validation erzielt werden kann. Zudem hat der OOB-Fehler auch den Vorteil, dass zur Schätzung dafür nur ein RF erzeugt werden muss, im Gegensatz dazu müssen bei einer k-fold *Cross-Validation k* RF erzeugt werden. D.h. der OOB-Fehler verfügt auch über eine schnellere Berechenbarkeit. Die RF werden solange trainiert, bis der OOB-Fehler sich stabilisiert.

## 3.6. Boosting

Die im vorigen Kapitel beschriebene Methode des *Baggings*, hilft bei der Reduzierung der Varianz. Da aber für das Aggregieren so viele *Trees* wie möglich erzeugt werden sollen, kann bei sehr großen Datenmengen die Rechenperformance stark nachlassen. Die Idee des *Boostings* greift genau an dieser Stelle ein und verfolgt den Ansatz viele *Weak-Learner* zu erzeugen und diese so zu kombinieren, dass der *Output* ähnlich wie bei den *Random Forests* die Varianz, und zusätzlich den Bias, verringert. Ein *Weak-Learner* sei im folgenden definiert als ein Klassifizierer, welcher nur leicht mit der echten Klassifizierung korreliert und *Strong-Learner* als jener welcher eine beliebig starke Korrelation mit der echten Klassifizierung hat. *Weak-Learner* sind z.B. stark reduzierte *Decision Trees*, welche aus nur einem einzigen Knoten und zwei *Outputs* bestehen, dieser wird auch als *Decision Stump* anstatt *Tree* bezeichnet. *Strong-Learner* sind im Zusammenhang mit *Decision Trees* z.B. *Random Forests*. *Boosting* wird oft auch als *Ensemble-Algorithmus* in *Supervised Learning* bezeichnet, denn er erzeugt aus einer Familie von *Weak-Learner* einen *Strong-Learner*.

### 3.6.1. AdaBoost

Die von Y. Freund und R. Schapire entwickelte Methode des *Adaptive Boosting*, kurz *AdaBoost*, verfolgt die Idee des *Boostings* folgendermaßen [18]: Sie verbessert die Leistung von Lernalgorithmen, indem sie Ergebnisse von den einzelnen *Weak-Learner* zu einer gewichteten Summe für die Klassifizierung kombiniert. *AdaBoost* passt die nachfolgenden *Learner* so an, dass sie sich auf falsch klassifizierte Instanzen konzentrieren, wodurch ein *Overfitting* verhindert werden kann. Trotz schwacher einzelner *Learner* konvergiert *AdaBoost* zu einem *Strong-Learner*, solange dieser zumindest etwas besser abschneidet, als wie bei zufälligem Raten. Er ist effektiv für die binäre Klassifizierung, kann aber auch auf mehrere Klassen erweitert werden [19]. Im Allgemeinen kann beim *Boosting* durch Addieren von beliebigen Basisfunktionen ein Klassifizierer gebildet werden. Diese Basisfunktionen haben die Form:

$$f(x) = \sum_{m=1}^{M} \alpha_m b(x; \gamma_m)$$

mit Koeffizienten $\alpha_m$ und $b(x; \gamma) \in \mathbb{R}$. Beim *Boosting* von *Trees* ist $\gamma$ die Parametrisierung von den *Split*-Variablen und Vorhersagen der Blätter. Damit diese Modelle auch eine Verbesserung beim Lernen in jedem Schritt haben, werden sie auf ein Minimum der zugrundeliegenden *Loss function* angepasst.

$$\min_{\alpha_m, \gamma_m} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \alpha_m b(x_i; \gamma_m))$$

Dieser Ausdruck ist jedoch in der Regel mit sehr hoher Rechenleistung verbunden und wird in der Praxis deshalb durch ein *Forward Stagewise Additive Modeling* angenähert [13]. Nach jedem Schritt $m$ wird die *Output Funktion* um einen additiven Term der Basisfunktion erweitert und anschließend wieder auf $\alpha_m, \gamma_m$ minimiert:

$$f_m(x) = f_{m-1}(x) + \alpha_m b(x; \gamma_m) \tag{3.7}$$

Für *AdaBoost* soll nun die *Exponential Loss Function* verwendet werden

$$L(y, f(x)) = \exp(-yf(x))$$

Damit gilt es für den Klassifizierer $G_m(x) \in \{-1, 1\}$ folgende Gleichung für jeden Schritt $m$ zu lösen:

$$\min_{\alpha_m, \gamma_m} = \arg\min_{\alpha,\gamma} \sum_{i=1}^{N} \exp(-y_i(f_{m-1}(x_i) + \alpha G(x_i))) \tag{3.8}$$

$$= \arg\min_{\alpha,\gamma} \sum_{i=1}^{N} w_i^{(m)} \exp(-y_i \alpha G(x_i)) \tag{3.9}$$

mit einem Gewicht $w_i^{(m)} = \exp(y_i f_{m-1}(x_i)$, welches unabhängig von $\beta$ und $G(x)$, aber abhängig vom vorigen Schritt ist. Teilt man nun die Summe auf $y_i = G(x_i)$ und $y_i \neq G(x_i)$ auf und betrachtet nur den von $G$ abhängigen Term, so erhält man:

$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

mit diesem $G_m$ lässt sich nun auch das $\alpha_m$ bestimmen und zwar ergibt sich durch Einsetzen in (3.8)

$$\alpha_m = \frac{1}{2} \log \frac{1 - err_m}{err_m}$$

mit einer gewichteten Fehlerrate

$$err_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))}{\sum_{i=1}^{N} w_i^{(m)}} \tag{3.10}$$

Dadurch lässt sich mit (3.7) die Funktion für den nächsten Schritt ermitteln und durch

$$w_i^{(m+1)} = w_i^{(m)} \exp(-\alpha_m y_i G_m(x_i))$$
$$= w_i^{(m)} \exp(2\alpha_m I(y_i \neq G_m(x_i))) \exp(-\alpha_m)$$

lassen sich auch die Gewichte neu berechnen. Beachte, dass dabei der Term $\exp(-\alpha_m)$ am Ende vernachlässigt werden kann, da alle Gewichte mit dem selben Faktor skaliert werden. Des Weiteren wird in der Literatur meistens der Faktor $2\alpha_m = \beta_m$ ersetzt.

Zur Bestimmung eines binären Klassifizierers (VN kauft oder kauft nicht) $\hat{G}(F_i)$ für eine Outpout Variable $C_i \in \{-1, 1\}$ mit Kundenfeatures $F_{i,n}$, gilt somit (siehe Abbildung 3.5 für ein Beispiel auf dem Iris Testdatensatz):

**Algorithm 9** AdaBoost M1

**Require:** Trainingsmenge $Y \subseteq \Omega = (F, C)$

**Ensure:** AdaBoost Klassifizierer

1: Initialisiere die Gewichte $w_i = \frac{1}{I}, i = 1, \ldots, I$

2: **while** $1 \leqslant m \leqslant M$ **do**

3: Passe $G_m(F_i)$ an die Trainingsdaten mit Gewicht $w_i$ an

4: Berechne die gewichtete Fehlerrate:

$$err_m = \frac{\sum_{i=1}^{I} w_i^{(m)} I(C_i \neq G(F_i))}{\sum_{i=1}^{I} w_i^{(m)}}$$

5: Berechne:

$$\beta_m = \log \frac{1 - err_m}{err_m}$$

6: Berechne:

$$\beta_m = \log \frac{1 - err_m}{err_m}$$

7: Aktualisiere die Gewichte:

$$w_i \leftarrow w_i \exp(\beta_m I(C_i \neq G_m(F_i))), i = 1, \ldots, I$$

8: **end while**

9: Output:

$$\hat{G}(F_i) = sign\left(\sum_{m=1}^{M} \beta_m G_m(F_i)\right)$$

### 3.6.2. Boosting Trees

Anstatt wie bei *AdaBoost* nur auf *Stumps* zu trainieren, können die *Boosting*Methoden auch auf *Decision Trees* als *Weak-Learner* erweitert werden, diese Vorgehensweise wird dann als *Boosting Trees* bezeichnet. Es hat sich gezeigt, dass das *Boosting* von *Trees* weitaus bessere Leistungen erbringt als einzelne *Stumps* [20]. Sei $R_m$ wieder die Regionen der zu betrachtenden *Decision Trees*, im Knoten $m$ wie aus (3.6), dann weist der *Decision Tree* (hier formell als Funktion $f$) jedem Beobachtungswert $x$ eine Konstante $\gamma_m$ zu:

$$x \in R_m \Rightarrow f(x) = \gamma_m$$

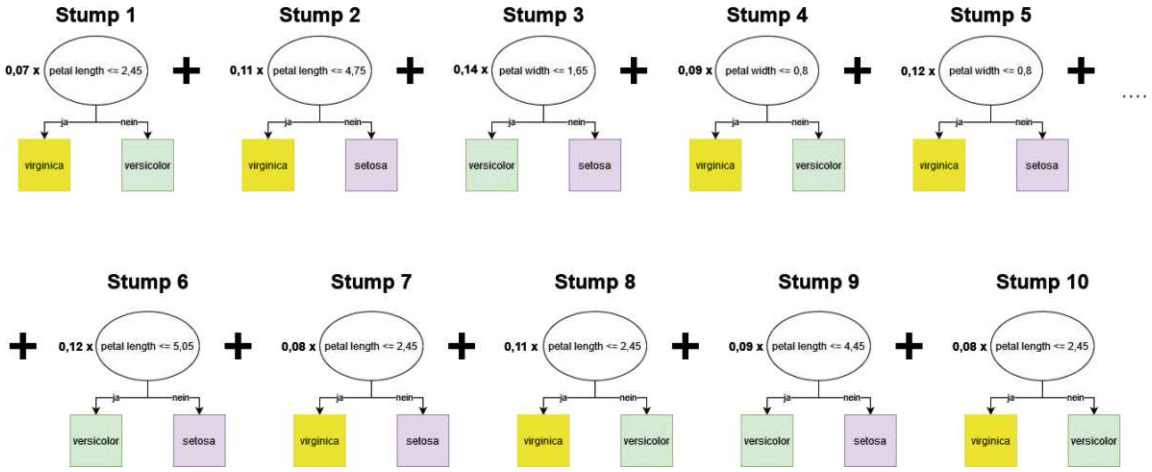


Abbildung 3.5.: Beispiel eines AdaBoost Klassifizierer mit 10 Stumps für den Iris-Testdatensatz

Damit kann das *Boosting* von *Trees* mit $\mathcal{R} = \{R_m, \gamma_m\}$ formuliert werden als:

$$T(x; \mathcal{R}) = \sum_{m=1}^{M} \gamma_m I(x \in R_m)$$

$\mathcal{R}$ soll nun so optimiert werden, dass der *Loss* zu den tatsächlichen Beobachtungen $y_i$ minimiert wird

$$\hat{\mathcal{R}} = \arg\min_{\mathcal{R}} \sum_{m=1}^{M} \sum_{x_i \in R_m} L(y_i, \gamma_m)$$

Da der *Boosted Tree* eine Summe von den *Trees* $T(x; \mathcal{R})$ ist muss, wie bei Kapitel 3.6.1 mit dem *Forward Stagewise Additive Modeling*, bei jedem *Forward Step* eine Optimierungs-Lösung für $\{R_m, \gamma_m\}$ gefunden werden.

$$\hat{\mathcal{R}}_m = \arg\min_{\mathcal{R}_m} \sum_{x_i \in R_{i,m}} L(y_i, f_{m-1}(x_i) + T(x_i; \mathcal{R})) \tag{3.11}$$

$$\hat{\gamma}_{i,m} = \arg\min_{\gamma_{i,m}} \sum_{x_i \in R_{i,m}} L(y_i, f_{m-1}(x_i) + \gamma_{i,m}) \tag{3.12}$$

Eine Methode, die diese Optimierungsaufgabe effizient und für allgemeine differenzierbare *Loss Functions* löst, und sich auch in der Praxis bewährt hat ist die numerische Optimierung durch ein Gradientenverfahren.

### 3.6.3. Gradient Boosting

*Gradient Boosting* oder *Gradient Boosted Decision Trees* (GBDT) basiert auf dem Konzept des *Boosting* und verallgemeinert diesen Ansatz auf beliebige differenzierbare *Loss Functions* [21]. Im Kontext des *Gradient Boosting* sind die Residuen die Unterschiede zwischen

den beobachteten Zielwerten und den Vorhersagen des aktuellen Modells. Durch iterative Minimierung dieser Residuen verfeinert der Algorithmus schrittweise seine Vorhersagen und konvergiert zu einem möglichst genauen Modell. Die hierfür verwendete Methode zur Minimierung ist die *Gradient Descent Optimierung*. Durch diesen Ansatz kann *Gradient Boosting* mit einem breiten Spektrum von *Loss Function* umgehen, was es vielseitig und für verschiedene Datentypen und Probleme anwendbar macht. Bei jeder Iteration des *Gradient Boosting Algorithmus* wird ein neuer *Weak-Learner* trainiert, um die Residuen des vorherigen Modells vorherzusagen. Die Vorhersagen aller *Weak-Learner* werden dann zu einer gewichteten Summe kombiniert, wobei das Gewicht jedes Learners durch seinen Beitrag zur Reduktion der Gesamtverlustfunktion bestimmt wird. Durch das iterative Hinzufügen neuer *Weak-Learner* und die Optimierung ihrer Parameter basierend auf dem Gradienten der *Loss Function* lernt *Gradient Boosting* effektiv komplexe Beziehungen innerhalb der Daten und erzeugt genauere Vorhersagen, als wie *AdaBoost* [22]. Die Fähigkeit des Algorithmus, mit verschiedenen *Loss Functions* umzugehen, und seine Flexibilität bei der Modellauswahl machen ihn zu einer beliebten Wahl für ein breites Spektrum von Aufgaben des maschinellen Lernens.

Um nun (3.11) numerisch lösen zu können, wird bei Schritt $m$ der zu optimierende Parameter in die Richtung $h_m = -\rho_m g_m$ verbessert. Dabei ist $\rho_m \in \mathbb{R}$ die Schrittweite und $g_m \in \mathbb{R}^N$ sei der Gradient von der zugrundeliegenden Loss Function $L(f)$:

$$g_{i,m} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} \tag{3.13}$$

$$\rho_m = \arg\min_{\rho} L(f_{m-1} - \rho g_m) \tag{3.14}$$

Damit kann anschließend die Funktion $f$ für den nächsten Schritt $m$ aktualisiert werden

$$f_m = f_{m-1} - \rho_m g_m$$

und erneut die Koeffizienten für den zu optimierenden Parameter berechnet werden.

*Forward Stagewise Boosting* erzeugt *Decision Trees* welche (3.11) maximal reduziert. Also sind die Vorhersagen für den *Tree* analog zu den Komponenten des negativen Gradienten (3.13). Der Hauptunterschied zwischen ihnen besteht darin, dass die Komponenten im Tree nicht unabhängig sind. Wäre nun die Minimierung des Verlustes auf dem Trainingsdatensatz das einzige Ziel, würde der *Gradient Descent* die bevorzugte Strategie sein. Der Gradient (3.13) ist trivial für jede differenzierbare Verlustfunktion $L(y, f(x))$ zu berechnen. Weiters ist der Gradient nur für die Trainingsdatenpunkte $x_i$ definiert, während das eigentliche Ziel darin besteht, $f_M(x)$ auf neue Daten zu verallgemeinern, die nicht in der Trainingsmenge enthalten sind. Ein möglicher Ausweg aus diesem Dilemma ist die Induktion eines Trees $T(x; \tilde{\mathcal{R}}_m)$ , dessen Vorhersagen so nahe wie möglich am negativen Gradienten liegen. Wenn man den quadratischen Fehler zur Messung der Nähe verwendet, führt dies zu

$$\tilde{\mathcal{R}}_m = \arg\min_{\mathcal{R}} \sum_{n=1}^{N} (-g_{n,m} - T(x_n; \mathcal{R}))^2 \tag{3.15}$$

Auch wenn die durch (3.15) erzeugten Regionen $\tilde{R}_{j,m}$ sich von den eigentlichen Regionen $R_{j,m}$ aus (3.11) unterscheiden, sind sie jedoch immer noch ähnlich genug [13].

Eine Regularisierung in *Gradient Boosting Decision Trees* ist wichtig, um ein *Overfitting* zu verhindern und das Modell so allgemein wie möglich zu belassen. Dazu gehören Techniken wie *Shrinkage*, *Pruning* und *Subsampling* von Merkmalen. Diese Methoden steuern die Komplexität des Modells und fördern ein robusteres Lernen aus den Daten, was zu einer besseren Leistung bei neuen untrainierten Daten führt. Die einfachste Methode einen *Shrinkage Parameter* bei den GBDT einzuführen, ist ein Lernparameter $0 < \nu < 1$ einzuführen und bei der Aktualisierung von $f_m$ zu multiplizieren

$$f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J} \gamma_{j,m} I(x \in R_{j,m})$$

Es ist leicht zu sehen, dass kleine Werte für $\nu$ den Test-Fehler besser annähern und zugleich mehrere *Boost*-Iterationen benötigen. In der Praxis erwies sich $\nu < 0.1$ und *Boost*-Schritte $M$ mit einem vorzeitigen Stopp.
Die Idee bei *Subsampling* ist, wie in 3.5.1 durch *Bagging* die Varianz eines ungenauen Klassifizierer zu verringern. Angewandt auf die GBDT wurde ein *Stochastic Gradient Boosting* hergeleitet [23]. Beim stochastischen *Boosting* wird bei jeder Iteration eine Teilmenge der Trainingsdaten nach dem Zufallsprinzip ausgewählt, wodurch Stochastizität in den Algorithmus eingeführt wird. Diese Stochastik trägt dazu bei, ein *Overfitting* zu verhindern und die Berechnungszeit zu verkürzen, wodurch stochastisches *Boosting* besonders effektiv für große Datensätze ist.

**Algorithm 12** Gradient Boosting Decision Trees

**Require:** Trainingsmenge $Y \subseteq \Omega = (\mathcal{F}, C)$; Loss Funktion $L(C, f(\mathcal{F}))$

**Ensure:** Boosted Decision Tree Klassifizierer

1: Initialisiere:

$$f_0 = \arg\min_{\gamma} \sum_{i=1}^{I} L(C_i, \gamma)$$

2: **while** $1 \leqslant m \leqslant M$ **do**

3: Berechne Pseudo-Residuals für die zu optmimierende Richtung $h_m = -\rho_m g_m$für $i = 1, \ldots, I$:

$$\begin{aligned} g_{i,m} &= \left[\frac{\partial L(C_i, f(\mathcal{F}_i))}{\partial f(\mathcal{F}_i)}\right]_{f(\mathcal{F}_i)=f_{m-1}(\mathcal{F}_i)} \\ \rho_m &= \arg\min_{\rho} L(f_{m-1} - \rho g_m) \end{aligned}$$

4: Trainiere einen Tree auf den negativen Gradienten mit quadratischem Loss:

$$\tilde{\mathcal{R}}_m = \arg\min_{\mathcal{R}} \sum_{i=1}^{I} (-g_{i,m} - T(\mathcal{F}_i; \mathcal{R}))^2$$

5: Berechne den Faktor $\gamma_m$:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{I} L(C_i, f_{m-1}(\mathcal{F}_i) + \gamma h_m(\mathcal{F}_i)$$

6: Aktualisiere:

$$f_m(\mathcal{F}) = f_{m-1}(\mathcal{F}) + \nu\gamma_m h_m(\mathcal{F}$$

7: **end while**

8: Output: $g_M(F)$

### 3.6.4. XGBoost

Eine skalierbare und effiziente Erweiterung der GBDT ist der *Extreme Gradient Boosting* (XGBoost), welche in den letzten Jahren zahlreiche *Machine Learning* Wettbewerbe gewonnen hat [24]. Die Idee des *XGBoost* ist es, die Rechenschritte des *Gradient Boosting* so weit wie möglich zu approximieren und die Rechenkapazitäten der CPU und RAM bestmöglich zu nutzen. Wie in der Originalveröffentlichung von T. Chen und C. Guestrin [25] beschrieben, besteht *XGBoost* aus mehreren Schritten:

1. Erstelle einen *Gradient Boost Decision Tree*
2. Regularisiere den Output von 1 durch *Shrinkage* und *Column Subsampling*
3. *Approximate Greedy Algorithmus*

4. *Weighted Quantile Sketch*
5. *Parallel Learning*
6. *Sparsity-aware Split Finding*
7. *Cache-aware Access*
8. *Out-of-core Computation*

Die Schritte 1 bis 2 sind die in Kapitel 3.6.3 beschriebenen Methoden zur Erzeugung eines regularisierten GBDT. Die letzten 6 Schritte 3 bis 8 dienen zur Optimierung für *Big Data*. Der *Approximate Greedy Algorithmus* trägt zu dieser Optimierung bei, indem er die Merkmale im Trainingsdatensatz nach Quantilen aufteilt. Danach werden diese in *Buckets* aggregiert und darauf werden die besten *Splits* für den *Tree* gefunden. Um die Performance der Quantile zu verbessern, werden die Quantile mit den *Hessians* $h$ gewichtet um Datenpunkte zu splitten, welche eine niedrige Vorhersagegenauigkeit aufweisen. Die Gewichte für *Split-Points* für das $k$-te Merkmal $\{s_{k,1}, \ldots, s_{k,l}\}$ sind also gegeben durch:

$$r_k(z) = \frac{1}{\sum_{(x,h)\in R_k}} \sum_{(x,h)\in R_k, x<z} h$$

Um für jedes Feature bei jeder Iteration den besten *Split* zu finden, benötigt der *Approximate Greedy Algorithmus* eine vorher sortierte Datenmenge. Das Sortieren ist mitunter der zeitaufwändigste Teil bei *Decision Trees*. Um diesen Nachteil zu reduzieren, speichert *XGBoost* die sortierten Daten in speicherinternen Einheiten, um diese jederzeit ohne Neuberechnung abrufen zu können. Als Nächstes werden aus der Trainingsdatenmenge, diejenigen herausgenommen, welche fehlende Werte in den Merkmalen haben. Auf der 'nicht-leeren' Menge wird der GBDT trainiert und die 'leere' Datenmenge wird auf beide Seiten durchberechnet und jener *Gain*, welcher eine Klasse $k$ maximiert ist die neue Klassifizierung für diesen Datenpunkt. Damit beim Finden der *Split-Points* die CPU nicht mit unnötigen Lesen/Schreiben Abhängigkeiten zu kämpfen hat, werden die Berechnungen der Gradienten und Hessians explizit einem internen *Buffer* zugeordnet. Um die Berechnung des *XGBoost* Algorithmus noch weiter zu optimieren werden sogennante *Out-of-Core* Berechnungen durchgeführt. Eines davon ist die Block Compression, zum Komprimieren der Blöcke nach Spalten. Die zweite Methode ist das *Block Sharding*, dieses nutzt das parallele Lesen/Schreiben von mehreren *Hard-Disks* aus und teilt die Datenmenge auf diese auf.

### 3.6.5. LightGBM

Obwohl in *XGBoost* viele technische Optimierungen eingeflossen sind und sich der Algorithmus im letzten Jahrzehnt bewährt hatte, gibt es dennoch einzelne Kritikpunkte. Nämlich ist die Effizienz und Skalierbarkeit für *Big Data* mit einer hohen Dimension des Merkmalraumes immer noch ausbaufähig. Der von Microsoft entwickelte *LightGBM* ist eine erfolgreiche Umsetzung und Erweiterung für *Gradient Boosting Algorithmen*, welcher den Schwachpunkt von GBDT numerisch vereinfacht und beschleunigt. Einer der Hauptgründe wieso GBDT so aufwändig zum Berechnen ist, ist dass für jedes Merkmal alle

Datenpunkte durchlaufen werden müssen, die *Trees* werden *depthwise* generiert. Um dieses Problem zu lösen, sollen die Anzahl der Datenpunkte und Merkmale gewichtet werden und vernachlässigbare Instanzen unberücksichtigt bleiben. Die Autoren Guolin Ke et. al. [26] führen dafür zwei Methoden ein. Zum einen das *Gradient-based One-Side Sampling* (GOSS) und zum anderen das *Exclusive Feature Bundling* (EFB). Der durch GOSS und EFB erzeugte GBDT wird als *LightGBM* bezeichnet und erzielt eine ähnliche Genauigkeit bei einer bis zu 20-fachen schneller Trainingszeit.

Es wird angenommen, dass der *Gain* durch die Varianz nach dem *Splitting* gemessen wird. In der vorgestellten GOSS Methode wird zuerst die Trainingsmenge absteigend nach ihrem absoluten Gradienten sortiert und nur die ersten $a \times 100\%$ Datenpunkte als Untermengen $A$ behalten, die restliche Menge $A^c$ mit den $(1-a) \times 100\%$ kleinen Gradienten werden weiter geteilt auf eine Untermenge $B$ mit Größe $b \times |A^c|$. Damit kann nun nach der Varianz auf der Menge $A \cup B$ der beste *Gain* und damit der nächste *Split* berechnet werden. Die Varianz berechnet sich für den Punkt $d$ mit Merkmal $j$ und Gradienten $g_i$ für Links $l$ und Rechts $r$ wie folgt:

$$\tilde{V}_j(d) = \frac{1}{n}\left( \frac{(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i)^2}{n_l^j(d)} + \frac{(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i)^2}{n_r^j(d)} \right)$$

Damit schätzt GOSS die Varianz auf einer kleiner Menge als wie bei ursprünglichen GBDT Verfahren. Der dadurch entstehende Approximationsfehler $\mathcal{E}$ konvergiert für zunehmende Datenmengen gegen 0, d.h. die verwendete Approximation ist genau genug. Es sei auch angemerkt, dass der Fall mit $a = 0$ dem *Random Sampling* voriger Methoden entspricht. Weiters wird durch *Sampling* die Diversität der zugrundeliegenden *Weak-Learner* verbessert, was hilfreich für eine bessere Allgemeine Leistung ist [27].

Die zweite eingeführte Methode, EFB, soll bei hochdimensionalen Daten mit schwachbesetzten Datenpunkten eine Möglichkeit liefern, möglichst verlustfrei Merkmale zu vernachlässigen. Die Idee ist es Merkmale, welche sich gegenseitig ausschließen bzw. nie gleichzeitig nicht leer sind, in *Bundles* zu vereinen. Idealerweise mit $\#Bundle \ll \#Merkmal$ erzielt man somit eine signifikant schnellere Berechenbarkeit. Der EFB Algorithmus bündelt also den Merkmalraum auf eine niedrigere Dimension mit dichteren (nicht-leeren) Merkmalen. Weiters werden die Merkmale im selben *Bundle* zusammengefügt, um die Training-Komplexität zu reduzieren.

**Algorithm 15** LightGBM

**Require:** Trainingsmenge $Y \subseteq \Omega = (\mathcal{F}, C)$; Merkmale: $F$; Loss $L(C, f(\mathcal{F}))$; Big Gradient Sampling Ratio: $a$; Small Gradient Sampling Ratio: $b$

**Ensure:** LightGBM Klassifizierer

1: Kombiniere Merkmale nach der EFB-Methode

2: Initialisiere:

$$T_0 = \arg\min_{\gamma} \sum_{i=1}^{I} L(C_i, \gamma)$$

3: **while** $1 \leqslant m \leqslant M$ **do**

4: Berechne die absoluten Gradienten für $i = 1, \ldots, I$:

$$g_{i,m} = \left[ \frac{\partial L(C_i, T(\mathcal{F}_i))}{\partial T(\mathcal{F}_i)} \right]_{T(\mathcal{F}_i)=T_{m-1}(\mathcal{F}_i)}$$

5: Resampling nach der GOSS-Methode:

$$\begin{aligned} topN &= a \times length(Y); randN = b \times length(Y); sorted = sort(abs(g_{i,m})); \\ A &= sorted[1 : topN]; \\ B &= RandomPick(sorted[topN : length(Y)], randN); \\ \hat{Y} &= A \cup B; \end{aligned}$$

6: Berechne den Varianz als Information Gain:

$$\tilde{V}_{F_j}(d) = \frac{1}{n} \left( \frac{(\sum_{x_i \in A_l} g_i + \frac{1-a}{b} \sum_{x_i \in B_l} g_i)^2}{n_l^{F_j}(d)} + \frac{(\sum_{x_i \in A_r} g_i + \frac{1-a}{b} \sum_{x_i \in B_r} g_i)^2}{n_r^{F_j}(d)} \right)$$

7: Erzeuge einen neuen DT $\hat{T}_m(\mathcal{F})$ auf der Menge $\hat{Y}$

8: Aktualisiere

$$T_m(\mathcal{F}) = T_{m-1}(\mathcal{F}) + T_m(\mathcal{F}$$

9: **end while**

10: Output: $\tilde{T}(\mathcal{F}) = T_M(F)$

### 3.6.6. CatBoost

Nur zwei Jahre nach der Veröffentlichung von LightGBM von Microsoft, veröffentlichten Liudmila Prokhorenkova et. al. [28] eine weitere Optimierung des GBDT mit dem Fokus, Kategorische Merkmale besser einzubauen, und um das *Data Leakage* zu verringern. *Data Leakage* (Datenleck) tritt dann auf, wenn die Trainingsdaten Informationen über die Klassifzierung enthalten, aber ähnliche Daten nicht verfügbar sind, wenn das Modell für die Klassifizierung verwendet wird. Dadurch wird die Leistung im Training verstärkt.

Gleichzeitig wird es aber für neue Daten schlechter abschneiden. *CatBoost* führt zwei neue Methoden ein, um den durch *Data Leakage* enstehenden Klasssifizierungs-Shift zu verringern. Eine der Implementierungen ist das *Ordered Boosting* und die zweite eine spezielle Verarbeitung von Kategorischen Merkmalen.

*CatBoost* hat im Vergleich zu *XGBoost* und *LightGBM* eine eigene Methode um kategorische Merkmale zu verarbeiten. Bereits in der Version von *LightGBM* deuteten die Autoren darauf hin, dass es bei Kategorischen Merkmalen mit hoher Kardinalität (z.B. User ID) effizienter sei, diese in numerische umzuwandeln [26]. Aufgrund dessen wird bei *CatBoost* eine *Ordered Target Statistic* benutzt um die Merkmale mit mehr als zwei Optionen auf numerische Werte mit minimalen Informationsverlust zu verarbeiten. *Ordered Target Statistic*, inspiriert durch diverse *Learning Algorithmen*, baut auf dem Ordering Principle auf und kombiniert dies mit zufälligen Permutationen der Trainingsmenge für jeden Schritt beim Boosting. D.h., dass nach der Permutation $\sigma$ die einzelnen Zeilen der Trainingsmenge Schritt für Schritt durchlaufen und immer um eins erweitert werden. Aus diesen monoton wachsenden Teilmengen $X_k$ werden anschließend die *Target Statistic* berechnet:

$$\hat{x}_k^i = \frac{\sum_{x_j \in X_k} \mathbb{1}_{\{x_j^i = x_k^i\}} y_j + ap}{\sum_{x_j \in X_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

Das zuvor angesprochene *Data Leakage* bei GBDT's und der Beweis, dass unter der Verwendung der selben Trainingsmenge für jeden Schritt beim Boosting einen starken Bias erzeugt, ist die zentrale Aussage von Prokhorenkova et. al. [28]. Die Lösung zur Bekämpfung dessen, ist es bei jedem Schritt auf einer neuen unabhängigen Menge zu trainieren. Indem das vorherige Modell auf die neue Menge trainiert wird, erhält man dadurch Residuen ohne *Shift*. In der Praxis werden jedoch eine sehr große Menge an verschiedenen Modellen bzw. *Trees* benötigt. Deshalb verfolgt *CatBoost* einen modifizierten Ansatz des GBDT. Zum Einen sind die erzeugten *Weak-Learner* eine Art von Symmetrischen (oft auch *Oblivious*) Bäumen, d.h. die Knoten auf der selben Ebene beinhalten alle den selben *Split*, diese Art von Trees sind ausgeglichener, haben einen niedrigeren Drang zum *Overfitting* und sind zusätzlich schneller berechenbar als wie reguläre *Decision Trees*. Eine zweite Modifikation zum GBDT ist den Gradienten zur Optimierung der *Loss Function* mit der Kosinus-Ähnlichkeit zweier Vektoren anzunähern:

$$cos(\theta) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Darüber hinaus werden nicht alle $n$ Ordered Models gespeichert. Um die Rechenleistung stark zu reduzieren, werden bei CatBoost nur die $M'_{r,j}(i) = M_{r,2^j}(i)$ für $j = 1, \ldots, \lceil \log_2 n \rceil$ gespeichert und aktualisiert.

**Algorithm 18** CatBoost

**Require:** Trainingsmenge $Y \subseteq \Omega = (\mathcal{F}, C)$; Merkmale: $F$; Loss $L(C, f(\mathcal{F}))$; Lernrate $\alpha$
**Ensure:** CatBoost Klassifizierer

Erzeuge eine zufällige Permutation $\sigma(i)$
Initialisiere: $M_0(i) = 0$ für $i = 1 \dots n$
**while** $1 \leqslant j \leqslant \lceil \log_2 n \rceil$ **do**
    $M_{r,j}(i) = 0$ für $r = 1 \dots s, i = 1 \dots 2^{j+1}$
**end while**
**while** $1 \leqslant t \leqslant I$ **do**
    Erzeuge einen Oblivious Tree $T_t$ mit Splits nach der höchsten Kosinus-Ähnlichkeit
    Berechne den Gradienten von $M_0$
    **while** Leaf $j$ in $T_t$ **do**
        $b_j^t = -Durchschnitt\{Gradienten\}$ für $i = 1 \dots j$
    **end while**
    $M_0(i) = M_0(i) + \alpha b_{leaf(i)}$ für $i = 1 \dots n$
**end while**
Output: $C(F) = \sum_{t=1}^{I} \sum_j \alpha b_j^t \mathbb{1}_{\{leaf_j\}}$

## 3.7. Künstliche Neuronale Netze

In der sich ständig weiterentwickelnden Welt der künstlichen Intelligenz war die Entwicklung von künstlichen Neuronen bis hin zu *Deep-Learning* Techniken ein langer und von zahlreichenden bahnbrechenden Arbeiten geprägter Weg. Der Grundstein legten bereits 1943 von W. Pitts und W.n McCulloch, mit dem *McCulloch-Pitts-Neuron* [29]. Dieses wurde im Jahre 1957 von F. Rosenblatt, zum revolutionären *Perceptron-Modell* entwickelt[30]. Ein weiterer entscheidender Durchbruch war das Konzept der Backpropagation, der 1960 von H. J. Kelley [31] erläutert und 1962 von S. Dreyfus [32] weiter verfeinert wurde. Das 1986 von Rumelhart, Hinton und Williams geprägte Konzept der *Backpropagation* [33] entwickelte sich zu einem Eckpfeiler beim Training künstlicher neuronaler Netze und revolutionierte das Feld mit seiner Fähigkeit, komplexe Modelle durch iterative Fehlerkorrektur zu optimieren. S. Hochreiters Entdeckung des Problems des verschwindenden Gradienten im Jahr 1991 [34] unterstrich die inhärenten Hindernisse beim Training tiefer neuronaler Netze und drohte, den Fortschritt zu ersticken. Doch 2011 lieferten Y. Bengio, A. Bordes und X.r Glorot lieferten dafür einen Lösungsansatz unter Verwendung von *Rectified Linear Unit* (ReLU)-Aktivierungsfunktionen [35] und läuteten damit eine neue Ära der Effizienz und Effektivität des *Deep-Learning* ein. In letzter Zeit erhielten tiefe neuronale Netze viel Aufmerksamkeit für die Verwendung von *Recommender Systems*, wo CNNs und RNNs in verschiedenen Empfehlungskontexten Spitzenergebnisse erzielten.

Im Grunde sind künstliche neuronale Netze (KNN), welche sich an biologischen neuronalen Netzen orientieren, leistungsstarke Systeme für die parallele Verarbeitung. Sie ahmen die Struktur des Gehirns nach und bestehen aus Neuronen, die über Synapsen miteinander verbunden sind und die Stärke der elektrischen Impulse anpassen. Die Struktur eines KNN besteht typischerweise aus drei verschiedenen Arten von Ebenen, welche wiederum aus

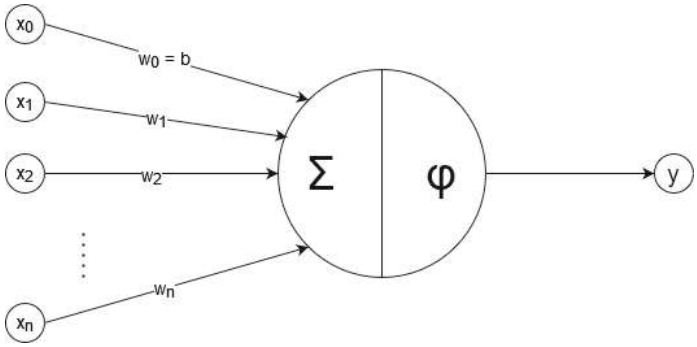


Abbildung 3.6.: Beispiel eines Perceptron mit $x_1, x_2, \ldots, x_n$ Eingabeparamter und $y$ als Output

mehreren künstlichen Neuronen bestehen. Die erste Ebene (*Layer*) besteht aus den Eingabeparameter $x_0, x_1, \ldots, x_n$ und wird als *Input Layer* bezeichnet. Die letzte Ebene besteht aus den Ausgabewerten $y_1, \ldots, y_n$ am Ende des Modellaufes und heißt *Output Layer*. Alle *Layer* zwischen der ersten und letzten Ebene sind die sogenannten *Hidden Layers*, welche die Neuronen im vorigen *Layer* mit einer gewichteten Summe und einer Aktivierungsfunktion $\varphi$ kombiniert und in den nächsten *Layer* einspeist. Eine fixe Anzahl der *Hidden Layer* ist nicht vorgegeben, jedoch steigt die Komplexität für jeden zusätzlichen *Layer* sehr stark.

### 3.7.1. Perceptron

Die einfachste Form eines KNN ist jene mit der geringsten Anzahl an *Hidden Layer* und künstlichen Neuronen, dieses Modell wurde von F. Rosenblatt entwickelt und besteht aus lediglich einem Neuron in einen einzigen *Hidden Layer*. 3.6

Dieses minimalistische KNN, auch *Perceptron* genannt, ist ein Algorithmus zum lernen eines binären Klassifikators. Die Werte der Eingabeparameter, gegeben durch $x_i \in \{0, 1\}$ für $i = 1, \ldots, n$ werden mit einer gewichteten Summe zusammengeführt und anschließend in eine Heaviside-Funktion (3.16) eingespielt.

$$\varphi(x) = \mathcal{H} = \begin{cases} 0, & \text{wenn } x < 0 \\ 1, & \text{wenn } x \geqslant 0 \end{cases} \tag{3.16}$$

Der Output lässt lässt sich also wie folgt formulieren:

$$y = \begin{cases} 0, & \text{wenn } \sum_{i=1}^{n} w_i x_i < b \\ 1, & \text{wenn } \sum_{i=1}^{n} w_i x_i \geqslant b \end{cases} \tag{3.17}$$

Die Grenze zur Klassifizierung $b$ und auch $x_0$ in (3.17) werden a priori festgelegt und dienen lediglich zur Unterscheidung der Resultate nach der Heaviside-Funktion. Der Hauptanwendungsfall eines *Perceptrons* ist die Klassifizierung zweier Variablen in einem linear separablen Raum (z.B. logische Funktionen wie AND und OR). Durch die einfache lineare Berechnung des *Perceptrons*, ist klar das die Anwendung auf leicht komplexere Funktionen, wie z.B. die Funktion XOR, nicht möglich sind bzw. mehrere gekoppelte Perceptrons

benötigt werden, aus diesem Grund wurde in weiterer Folge das *Perceptron Model* auf ein *Multilayer-Perceptron-Model* erweitert. Um nun das Netzwerk als einen *Strong-Learne* zu etablieren, reicht es nicht aus, die Neuronen mit einer einfachen linearen Aktivierungsfunktion wie der Heaviside-Funktion zu kombinieren. Das Sigmoid-Neuron ist eine brilliante Verknüpfung des *Perceptrons* mit der Sigmoid Funktion und ermöglicht somit eine Lernart ähnlich einer logistischen Regression (vgl. (3.4). Die Sigmoid Funktion ist gegeben durch

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

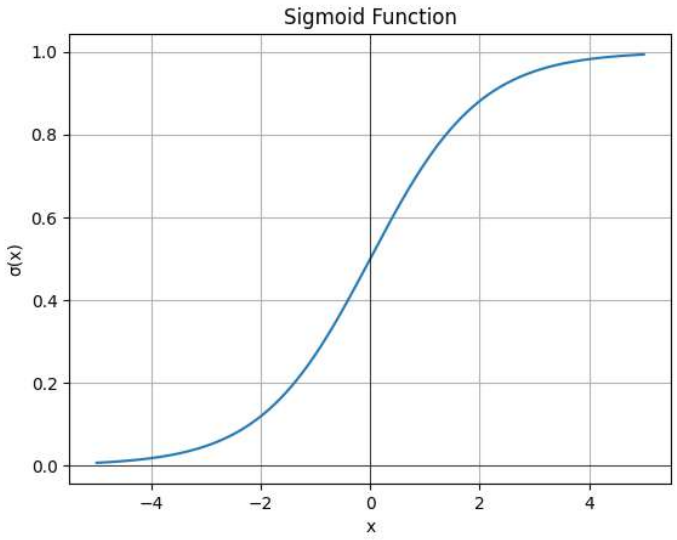


Abbildung 3.7.: Sigmoid Funktion

Damit ergibt sich durch Einsetzen der Output des Sigmoid-Neurons als

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b) = \frac{1}{1 + \exp(-\sum_{i=1}^{n} w_i x_i - b)}$$

Alleine durch die beiden Aktivierungsfunktionen ist das Perceptron in der Lage als Learner für sehr unterschiedliche Aufgaben zu dienen. Die richtige Wahl der Aktivierungsfunktion ist somit entscheidend für eine individuelle Aufgabe zur Klassifizierung. Eine Verallgemeinerung der Sigmoid Funktion auf einen $K$-dimensionalen Vektor ist die Softmax Funktion, mit Hilfe dieser Aktivierungsfunktion ist auch eine Klassifizierung in $K$ verschiedene Klassen möglich. Einige andere mögliche Aktivierungsfunktionen (vgl. 3.8 sind:

- Lineare Funktion
- Tanh
- Softsign: $SO(x) = \frac{x}{1+|x|}$

- ReLU (*Rectified Linear Unit*): $ReLU(x) = \max\{0, x\}$
- Parameterized ReLU ($\alpha > 0$): $PReLU(x) = \begin{cases} x, & \text{wenn } x > 0 \\ \alpha x, & \text{wenn } x \leqslant 0 \end{cases}$
- *Exponential Linear Unit* ($\alpha > 0$): $ELU(x) = \begin{cases} x, & \text{wenn } x > 0 \\ \alpha(e^x - 1), & \text{wenn } x \leqslant 0 \end{cases}$
- Swish: $SW(x) = x\sigma(\alpha x) = \frac{x}{1+e^{-\alpha x}}$

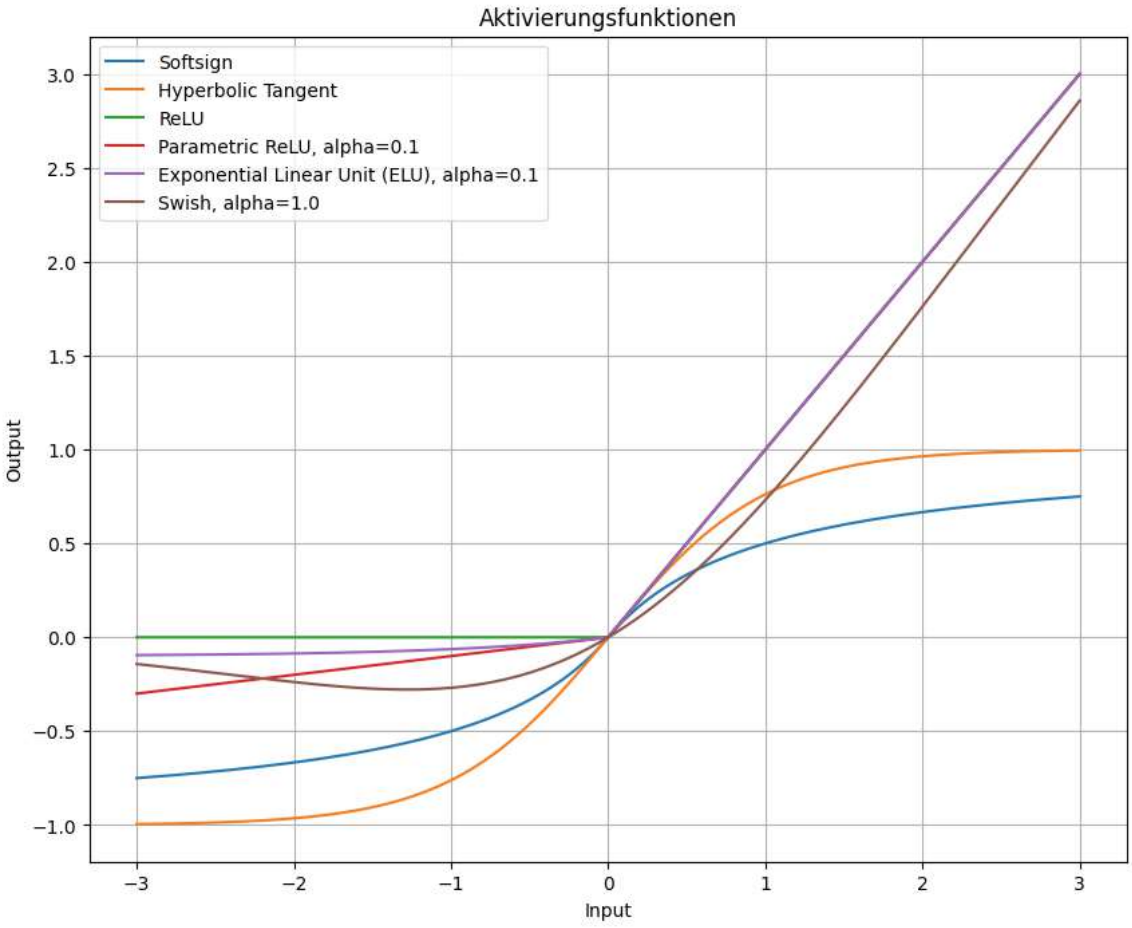


Abbildung 3.8.: Aktivierungsfunktionen

Damit ein *Perceptron* auf einer beliebig großen Datenmenge lernen kann und auch Schritt für Schritt näher der tatsächlichen Klassifizierung kommt, bedarf es eines Learning Algorithm welcher das Feedback des Klassifizierungs-Fehlers induktiv weitergibt. Die Motivation einen solchen Algorithmus einzuführen, folgt aus dem von Rosenblatt formulierten Approximations Satzes [30]. Damit ist die Konvergenz durch den *Perceptron Learning Algorithm* zu

einer Lösung nach einer endlichen Anzahl an Schritten möglich. Der Schritt des induktiven Lernen im *Learning Algorithm*, an dem die Gewichte $w_i$ angepasst werden, wird mit Hilfe der sogenannten *Delta Rule* durchgeführt. Die Fehlerfunktion (*Loss/Cost Function*) ist das Quadrat der Differenz zwischen den tatsächlichen Output in der Trainingsmenge und der vorhergesagten Klassifikation des Neurons. Der Fehler für den gesamten Trainingsdatensatz gegeben durch

$$\begin{aligned} E &= \sum_p E_p = \frac{1}{2}(y_p - \hat{y}_p)^2 \\ \hat{y}_p &= \sum_{i=1}^{n} w_i x_i + b \end{aligned}$$

Um diesen Fehler zu minimieren und die besten Gewichte für diese Trainingsmenge zu bekommen, wird die Methode des *Gradient Descent*, analog zu *Gradient Boosting* 3.6.3, angewandt. Die Veränderung der Gewichte wird also durch den Gradienten und einer Lernrate $\alpha$ bestimmt

$$\Delta_p w_i = -\alpha \frac{\partial E_p}{\partial w_i}$$

Durch Anwenden der Kettenregel und Einsetzen der Ableitung der Fehlerfunktion kann die Veränderung der Gewichte beschrieben werden mit

$$\Delta_p w_i = -\alpha (y_p - \hat{y}_p) x_{pi} \tag{3.18}$$

### 3.7.2. Feedforward Neural Network

Der Weg von den 1950er Jahren, mit dem *Perceptron Model*, bis heute hat eine unvorstellbare Vielfalt an Netzwerken mit mehreren Neuronen und auch mehreren *Hidden Layer* hervorgebracht. All diese bauen auf dem Prinzip des *Multi-Layer-Perceptrons* 3.9 auf und verknüpfen beliebig viele Neuronen mit den Eigenschaften ähnlich des Perceptrons.

In dieser Arbeit wird nur auf das Lernen eines *Feedforward Neural Network* näher eingegangen, da sich für die Klassifizierung von *Cross-Selling* mit der gegeben Datenstruktur am besten eignet. Für einen Überblick fast aller zur Zeit bestehenden Neuronalen Netzen veröffentlichte das Asimovinstitute 2019 eine Art *Cheatsheet* 3.10. Unabhängig für welche Aufgabe das NN am besten geeignet ist, gibt es im Allgemeinen zwei in der Struktur sehr verschiedene Kategorien: *Feedforward* und *Recurrent Neural Network*. Wenn der Informationsfluss des Netzes nur in eine Richtung (von *Input* nach *Output Layer*) geht, dann spricht man von einem *Feedforward Netz*. Bei einem FNN mit mehreren *Hidden Layers* wird auch von einem *Deep Neural Network* oder *Multi Layer Feedforward Neural Network* gesprochen. Ein FNN kann vollständig mit allen Neuronen verknüpft sein oder auch nur teilweise. Wichtig ist dabei das es keine Verbindungen zwischen Neuronen im selben *Layer* oder im vorigen *Layer* gibt.

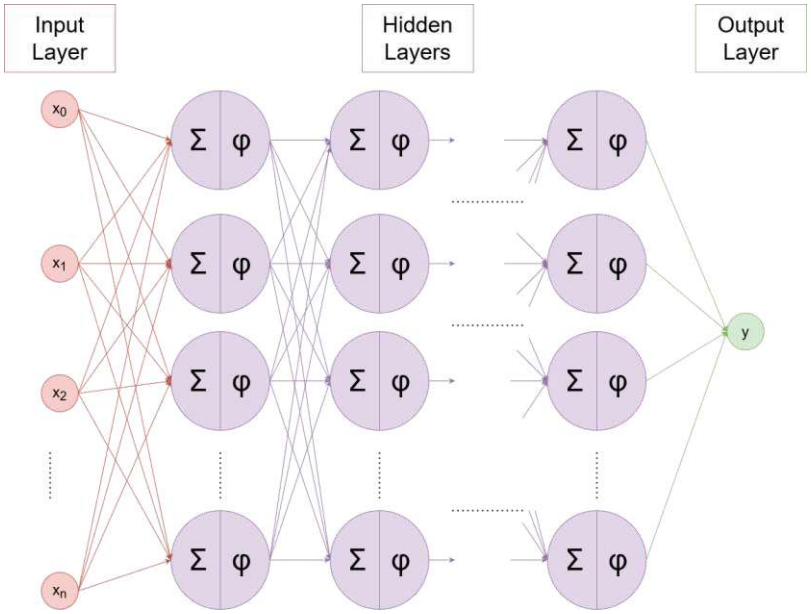


Abbildung 3.9.: Beispiel eines vollständig Verknüpften Multi Layer Perceptron bzw. Feed Forward Neural Network mit $x_0, x_1, x_2, \ldots, x_n$ Eingabeparamter und $y$ als Output

Wie am Anfang in Kapitel 3.7 beschrieben, haben NN ohne *Hidden Layer* starke Einschränkungen in ihrer Lernfähigkeit. Es wurde mehrfach gezeigt, dass NN mit bereits einem einzigen *Hidden Layer* in der Lage ist jede beliebige Funktion beliebig Nahe zu approximieren [36][**?**][37]. Das Resultat ist auch als *Universal Approximation Theorem* bekannt.

Sei nun $0 \leqslant l \leqslant L$ die Anzahl der *Layer*, dann bezeichnet $w_{ij}^{(l)}$ das Gewicht für die Verbindung vom Neuron $j$ im *Layer* $(l-1)$ zum Neuron $i$ im *Layer* $l$, dabei werden die Neuronen wie folgt indiziert: $i = 1, \ldots, d^{(l)}$ und $j = 1, \ldots, d^{(l-1)}$, wobei $d^{(l)}$ die Anzahl der Neuronen im *Layer* $l$ bezeichnet. Der *Input* Vektor $\vec{x}^{(l-1)} = (x_1^{(l-1)}, \ldots, x_{d^{(l-1)}}^{(l-1)})$ wird somit im *Layer* $l$ durch eine Summe mit der Gewichtsmatrix $W^{(l)}$ und dem Bias $b^{(l)}$ mit einer vorgegebenen Aktivierungsfunktion $\varphi$ verknüpft. Durch sukzessives Ausführen von *Layer* 1 bis *Layer* $L$ erhält man eine Output Funktion für das FNN:

$$FNN_{W,b} : \mathbb{R}^{d^{(0)}} \to \mathbb{R}^{d^{(L)}} \tag{3.19}$$

$$FNN_{W,b}(\vec{x}^{(0)}) = \varphi(W^{(L)}\varphi(\ldots\varphi(W^{(1)}\vec{x}^{(0)} + \vec{b}^{(1)})\ldots\vec{b}^{(L)}) \tag{3.20}$$

Durch diese Formulierung ist es nun möglich einen Lernprozess mit Hilfe einer Kostenfunktion

$$\mathcal{C}(W,b) = \frac{1}{N}\sum_{k=1}^{N}\mathcal{L}\left(F_{W,b}(\vec{x}_k), \vec{y}_k\right)$$

für ein Trainings-Datensatz $((\vec{x}_k, \vec{y}_k), k = 1, \ldots, N)$ zu definieren.

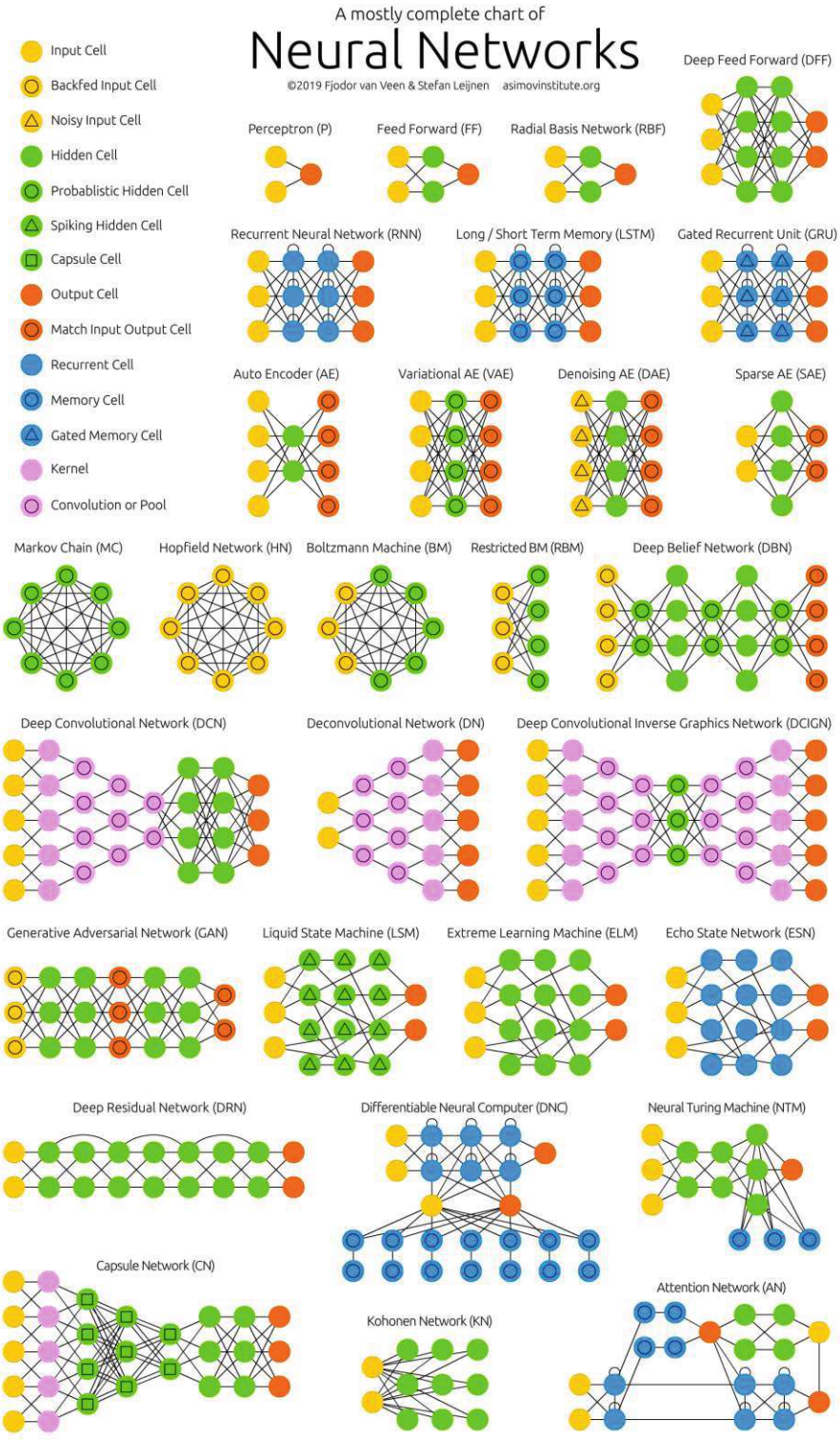


Abbildung 3.10.: Überblick einiger Neuronalen Netzwerke aus dem Jahr 2019 [38]

### 3.7.3. Kostenfunktionen

Beim *Learning Algorithm* für das *Perceptron* wurde als Kostenfunktion (*Cost Function*) das Quadrat der Differenz hergenommen, um daraus eine Veränderung der Gewichte im nächsten Iterationsschritt zu bewirken. Ähnlich wie bei den Aktivierungsfunktionen ist auch die Entscheidung der richtigen Kostenfunktion ein Hauptbestandteil zur Bildung eines FNN. Im Allgemeinen besteht eine Kostenfunktion aus einer Auswertung der Differenz zwischen dem tatsächlichen und dem berechneten Output, deshalb werden diese Art von Funktionen oft auch als Fehler- oder Verlustfunktionen (*Error-* oder *Loss Function*) bezeichnet. Einige Beispiele für Kostenfunktionen sind:

- Supremum Fehlerfunktion: $\mathcal{C}(W,b) = \sup_{x\in[0,1]} |F_{W,b} - y|$
- $L^2$-Fehlerfunktion: $\mathcal{C}(W,b) = ||F_{W,b} - y||_{L^2([0,1])}$
- Quadratische Fehlerfunktion: $\mathcal{C}(W,b) = \frac{1}{N}\sum_{K=1}^{N}(F_{W,b} - y)^2$
- Cross-Entropy: $\mathcal{S}(p,q) = \mathbb{E}^p[-\ln q(x)] = -\sum p(x)\ln q(x)$

Die *Cross-Entropy* Funktion, welche für das Trainieren von Klassifizierungsnetzwerke benutzt wird, ist zum Unterschied der anderen Kostenfunktionen eine Funktion zwischen zweier Wahrscheinlichkeitsdichten.

### 3.7.4. Backpropagation

Damit die Differenz der „tatsächlichen “Verteilung zum *Model Output* $F_{W,b}$ auch in die Veränderung der Gewichte einfließen kann, wurde beim *Perceptron* das Gradientenverfahren angewandt. Da es sich nur um ein Neuron handelte, war die Rückrechnung des Gradienten nicht weiter kompliziert. Nun muss für FNN eine ähnliche Rückrechnung der Gradienten über beliebig viele Layer und Neuronen hergeleitet werden. Diese Art des Lernens bei FNN wird wegen des Rückrechnens als *Backpropagation* Algorithmus bezeichnet. Der Rechenprozess kann durch sukzessives ausführen folgender Schritte beschrieben werden:

- Schritt 1: Das trainierte FNN macht eine Vorhersage $FNN_{W,b}$
- Schirtt 2: Der Unterschied zum tatsächlichen *Output* $y$ wird mit der gewählten Kostenfunktion berechnet
- Schritt 3: Der Fehler der Vorhersage wird durch *Backpropagation* mittels gewählten Optimierungsverfahren (z.B. Gradientenverfahren) weiter minimiert und die Gewichte und Bias in diese Richtung verändert

Sei nun auf einer Trainingsmenge von Versicherungsdaten $\mathcal{D} = \{(\vec{F}_k, \vec{C}_k), k = 1,\dots,N\}$ mit $\vec{F}_k, \vec{C}_k \in \mathbb{R}^I$ die Merkmale des VN und dessen Klassifizierung für ein Risikosegment $C_{kj} \in \{0,1\}, k = 1,\dots,N, j = 1,\dots,I$ a priori bekannt, dann sei der *Output* eines FNN gegeben durch (3.19). Weiters sei die Kostenfunktion gegeben durch

$$\mathcal{C}(W,b|\mathcal{D}) = \frac{1}{N}\sum_{k=1}^{N}\mathcal{L}(FNN_{W,b}(\vec{F}_k), \vec{C}_k)$$

Damit lässt sich nun das Optimierungsproblem formulieren:

$$\hat{\Theta} = \arg\min_{W,b} \mathcal{C}(W, b|\mathcal{D}) \tag{3.21}$$

Mit der Wahl des Gradientenverfahren und einer Lernrate $\alpha$ sollen somit die Parameter $\Theta = (W, b)$ wie folgt verändert werden:

- Start mit einem beliebigen Wert $\Theta^{(0)}$
- Berechnung der Gewichtsveränderung für den nächsten Schritt

$$\Theta^{(k+1)} = \Theta^{(k)} - \alpha \nabla_\Theta \mathcal{C}(\Theta^{(k)}$$

- Veränderung der Gewichte bis $\nabla_\Theta \mathcal{C}(\Theta^{(k)} \approx 0$

Die Herausforderung ist es nun die Ableitungen der Kostenfunktion $\frac{\partial \mathcal{C}}{\partial W_{ij}^{(l)}}, \frac{\partial \mathcal{C}}{\partial b_i^{(l)}}$ effizient zu berechnen. Eine rein numerische Approximation würde bei einem FNN mit hoher Anzahl an *Layer* und Neuronen eine enorme Rechenleistung benötigen und würde zudem Rechenfehler von *Layer* zu *Layer* mittragen. Eine elegantere Methode zur Berechnung der partiellen Ableitungen lieferten Rumelhart, Hinton und Williams [33], welche als *Backpropagation* bekannt wurde. Der mathematische Schlüsselpunkt in dieser Methode ist die Anwendung der Kettenregel für die Ableitungen über alle *Layer* hinweg. Sei nun der *Output* eines Neurons $i$ im *Layer* $l$ gegeben durch $z_i^{(l)} = W^{(l)}\vec{x} + b^{(l)}$ und der daraus resultierende Fehler $\delta_i^{(l)} = \frac{\partial \mathcal{C}}{\partial z_i^{(l)}}$ und in Summe $\delta^{(l)} = \nabla_{\vec{z}^{(l)}} C$. Das erste Ergebnis aus der *Backpropagation Methode* ist eine Gleichung für die Berechnung des letzten *Layer*

$$\delta_i^{(L)} = \frac{\partial \mathcal{C}}{\partial z_i^{(L)}} = \sum_{j=1}^{d^{(L)}} \frac{\partial \mathcal{L}}{\partial x_j^{(L)}} \frac{\partial x_j^{(L)}}{\partial z_i^{(L)}} = \frac{\partial \mathcal{L}}{\partial x_j^{(L)}} \varphi'(z_i^{(L)} \tag{3.22}$$

für die *Layer* $l = L-1, \ldots, 1$ gilt

$$\delta_i^{(l)} = \frac{\partial \mathcal{C}}{\partial z_i^{(l)}} = \sum_{j=1}^{d^{(l+1)}} \frac{\partial \mathcal{C}}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}}$$

wobei

$$z_j^{(l+1)} = \sum_{k=1}^{d^W} W_{jk}^{(l+1)} x_k^{(l)} + b_j^{(l+1)}$$

da in den *Hidden Layer* gilt, dass $x_k^{(l)} = \varphi(z_k^{(l)}$ erhält man die zweite Gleichung

$$\delta_i^{(l)} = \sum_{j=1}^{d^{(l+1)}} W_{ij}^{(l+1)} \delta_j^{(l+1)} \varphi'(z_i^{(l)}) \tag{3.23}$$

Damit fehlt nur noch eine Gleichung zur Berechnung von der partiellen Ableitung nach dem Bias $b$ und eine für die nach den Gewichten $W$. Für die dritte *Backpropagation* Gleichung, die Ableitung nach dem Bias, gilt:

$$\frac{\partial \mathcal{C}}{\partial \vec{b}^{(l)}} = \frac{\partial \mathcal{C}}{\partial \vec{z}_i^{(l)}} \frac{\partial \vec{z}_i^{(l)}}{\partial \vec{b}^{(l)}}$$

wobei $\vec{z}_i^{(l)} = \sum W_{ij}^{(l)} x_{ij}^{(l+1)} + b_i^{(l)}$ und damit die Ableitung nach $b_i^{(l)}$ konstant gleich 1.Und somit lautet die Gleichung:

$$\frac{\partial \mathcal{C}}{\partial \vec{b}^{(l)}} = \delta_i^{(l)} \tag{3.24}$$

Als letztes bleibt nur noch zu zeigen, wie die partielle Ableitung der Kostenfunktion nach der Gewichtsmatrix $W$ aussieht. Dafür gilt eine ähnliche Überlegung wie bei (3.24) und zwar dass $\vec{z}_i^{(l)}$ differenziert nach $W$ gleich dem *Input* $x_j^{(l-1)}$ aus dem vorherigen *Layer* $l-1$ ist. Somit gilt für die letzte *Backpropagation* Gleichung:

$$\frac{\partial \mathcal{C}}{\partial W_{ij}^{(l)}} = \frac{\partial \mathcal{C}}{\partial \vec{z}_i^{(l)}} \frac{\partial \vec{z}_i^{(l)}}{\partial W_{ij}^{(l)}} = \delta_i^{(l)} x_j^{(l-1)} \tag{3.25}$$

Sei nun wieder $\mathcal{D}$ die Trainingsmenge der VN, wie für (3.21) beschrieben, dann kann der *Backpropagation Algorithmus* wie folgt beschrieben werden.

**Algorithm 21** Backpropagation Algorithmus

**Require:** Trainingsmenge $\mathcal{D} \subseteq \Omega = (\mathcal{F}, C)$; Merkmale $\mathcal{F}$; Loss $L(C, f(\mathcal{F}))$; Kostenfunktion $\mathcal{C}$; Lernrate $\alpha$

**Ensure:** Parameter Optimierer

1: Initialisiere: $\mathcal{F} = \vec{x}$ und wähle Startpunkte $\Theta^{(0)} = (W, b)$
2: **while** $\nabla \mathcal{C}_\Theta \not\approx 0$ **do**
3: **for** $l = 1, \ldots, L$, berechne Forwardpropagation **do**
4: $\vec{x}^{(l)} = \varphi(z^{(0)}$
5: $\vec{z}^{(l)} = W^{(l)} \vec{x}^{(l-1)} + \vec{b}^{(l)}$
6: **end for**
7: Berechne den Output Fehler aus (3.22): $\delta_i^{(L)} = \frac{\partial \mathcal{L}}{\partial x_j^{(L)}} \varphi'(z_i^{(L)}$
8: **for** $l = L-1, \ldots, 1$, berechne Backpropagation **do**
9: Berechne (3.23): $\delta_i^{(l)} = \sum_{j=1}^{d^{(l+1)}} W_{ij}^{(l+1)} \delta_j^{(l+1)} \varphi'(z_i^{(l)})$
10: **end for**
11: Berechne die partiellen Ableitungen aus (3.24) und (3.22): $\nabla \mathcal{C}_{(W,b)} = \delta_i^{(l)} x_j^{(l-1)} + \delta_i^{(l)}$
12: nächste Wahl der Parameter: $\Theta^{(k+1)} = \Theta^{(k)} - \alpha \nabla \mathcal{C}_\Theta(\Theta^{(k)})$
13: **end while**
14: Output: $\hat{\Theta} = (\hat{W}, \hat{b})$

Durch diesen Algorithmus gibt es eine systematische Herangehensweise zur Optimierung der Parameter für ein FNN. Dennoch ist auch diese Optimierungsmethode zur Lösung von

(3.21) im Allgemeinen keine Garantie die tatsächlich optimalen Parameter zu finden. Ein einfaches Beispiel sei in Abbildung 3.11 aufgezeigt. Vor allem ist eine Region der Kostenfunktion, bei dem $\nabla \mathcal{C}_\Theta = 0$ gilt, sehr problematisch und der Algorithmus 20 erreicht dabei kein Ende, da die Iterationsschritte von Mal zu Mal kleiner werden. Zusätzlich kann auch die Ableitung der Aktivierungsfunktion verschwinden und verdeutlicht somit noch stärker das Problem von Plateaus.

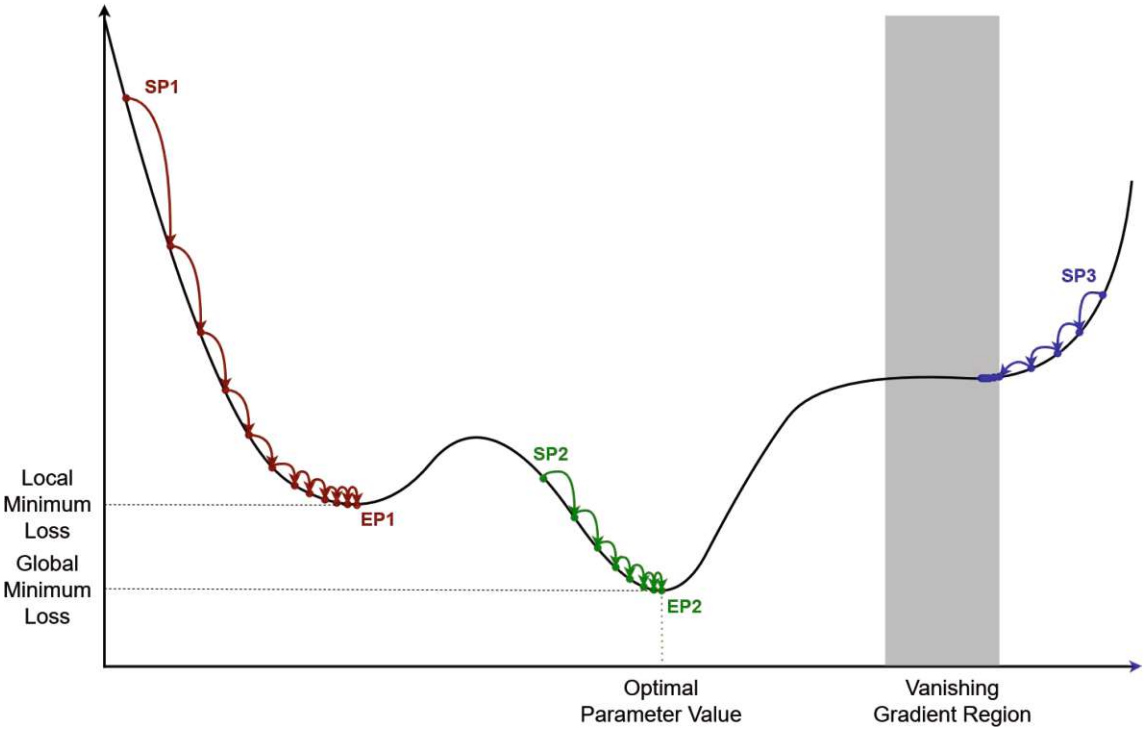


Abbildung 3.11.: Ein Beispiel einer Loss Funktion mit einem Plateau bei dem der Gradient verschwindet, einem lokalen und einem globalen Minimum. Nur der Punkt SP2 erzielt den tatsächlichen Optimalen Wert. Der SP1 erreicht sein Ende bei EP1, dem ersten gefundenen Minimum. Und der Punkt SP3 läuft in ein Plateau mit verschwindenen Gradienten und bildet einen Häufungspunkt von Enden.

### 3.7.5. Herausforderungen zur Optimierung von FNN

Wie am Ende des vorigen Kapitel beschrieben, gibt es für FNN im Allgemeinen keine Garantie das das globale Minimum der Kostenfunktion je erreicht wird. Wie sich heruasgestellt hat, gibt es für Neuronale Netze mehrere Optimierungsprobleme welche in den letzten Jahrzehnten starke Aufmerksamkeit auf sich zogen. In diesem Unterkapitel werden einige davon beschrieben und ein Lösungsvorschlag dazu formuliert.

### Lähmung des Netzes

Ein Problem, das beim Training eines neuronalen *Feedforward-Netzes* auftreten kann, ist die Netzlähmung (*Network Paralysis*). Dieses Problem tritt auf, wenn der *Backpropagation Algorithmus* einen stationären Zustand erreicht, was häufig auf extreme Werte der Gewichtsmatrix zurückzuführen ist. Wenn die Gewichte zu hoch werden, können sie zu sehr hohen *Inputs* für einige *Hidden Neuronen* führen. Bei sigmoiden Aktivierungsfunktionen führt dies zu Ausgangswerten nahe 1, wodurch das Delta sehr nahe bei Null liegt. Infolgedessen werden die mit dem *Backpropagation Algorithmus* 20 berechneten Änderungen der Gewichte vernachlässigbar, wodurch der Lernprozess zum Stillstand kommt. Ähnlich verhält es sich, wenn die Gewichte sehr niedrig sind, so dass die Inputs einiger Neuronen sehr klein ist und die *Outputs* gegen Null gehen. Dies wiederum führt zu vernachlässigbaren Delta-Werten und minimalen Änderungen der Gewichte. Einen Lösungsvorschlag lieferten Wang et al. (2004) [39] und schlugen vor, die Aktivierungsfunktion für jedes Trainingsmuster zu variieren und die Aktivierungsfunktionen während des Trainings kontinuierlich anzupassen, um eine Sättigung zu vermeiden.

### Lokale Minima

Das Training eines neuronalen Netzes kann auch durch das Vorhandensein von lokalen Minima (3.11) behindert werden. Bei neuronalen FNN mit mehreren *Hidden Layer* ist diese Oberfläche oft komplex und weist mehrere Minima auf. Die Gradient Descent Methode kann somit auch zu einem lokalen Minimum konvergieren, das nicht das globale Minimum ist. Dieses Problem wird durch die Topologie des neuronalen Netzes im Verhältnis zur Komplexität des Problems beeinflusst. Zu komplexe Netze können unter einer hohen Varianz leiden, was zu unterschiedlichen Lösungen für verschiedene Trainingsmuster führt, während zu einfache Netze unterparametrisiert und verzerrt sein können. Eine Methode zur Lösung dieses Problems ist das *Weight Jogging*, bei dem kleine zufällige Änderungen am Gewichtsvektor vorgenommen werden und der Trainingsprozess fortgesetzt wird. Dies kann dem Algorithmus helfen, lokale Minima zu umgehen und möglicherweise zu einem globalen Minimum zu konvergieren. Globale Suchalgorithmen, wie z. B. *Simulated Annealing*, sind jedoch effektiver im Umgang mit lokalen Minima.[40] [41] [42]

### Nicht eindeutige Lösungen

Selbst wenn eine Lösung einem (lokalen oder globalen) Minimum entspricht, ist sie in manchen Fällen nicht eindeutig. Es kann mehrere Gewichtskombinationen oder Netztopologien geben, die zu demselben Wert der Kostenfunktion führen. Dies kann aufgrund unterschiedlicher Startpunkte geschehen, die zu unterschiedlichen, aber gleichwertigen Lösungen führen, oder aufgrund von überparametrisierten Netzen, bei denen die Komplexität der Architektur die Anforderungen der Anwendung übersteigt, wodurch flache Minima entstehen. Dieses Problem kann durch *Pruning Methoden* die Komplexität des Netzes durch Eliminierung unnötiger Gewichte oder Einheiten reduzieren. Eine Möglichkeit ist durch Informationen der zweiten Ableitung ein Abwägen zwischen Komplexität und Fehler miteinzubeziehen [43][44].

### Overfitting

Ein *Overfitting* liegt vor, wenn ein neuronales Netz zu viele Gewichte hat und sich zu eng an die Trainingsdaten anpasst, wodurch Rauschen erfasst wird und eine schlechte Generalisierung auf neue Daten erfolgt. Dies geschieht häufig, wenn das Modell auf das globale Minimum der Fehlerfunktion trainiert wird, ohne dass eine geeignete Regularisierung erfolgt. Regularisierungstechniken, wie z. B. Gewichtsabnahme, können helfen, eine Überanpassung zu verhindern. Gewichtsabnahme fügt der Fehlerfunktion einen Malus hinzu, der die Gewichte gegen Null schrumpfen lässt. Eine weitere wirksame Methode ist das frühzeitige Stoppen, bei dem das Training vor Erreichen des globalen Minimums abgebrochen wird. Die Verwendung eines Validierungsdatensatzes hilft bei der Bestimmung des optimalen Abbruchpunktes.

### Skalierung der Eingaben

Die Skalierung der Eingaben kann die Qualität der endgültigen Lösung erheblich beeinflussen. Wenn die Eingaben nicht standardisiert sind, können die Gewichte in der unteren Schicht ineffektiv skalieren, was zu einer suboptimalen Leistung führt. Die Standardisierung der Eingaben auf einen Mittelwert von Null und eine Standardabweichung von Eins stellt sicher, dass alle Eingaben gleich behandelt werden und ermöglicht einen sinnvollen Bereich für zufällige Startgewichte. Mehr dazu in Kapitel 4.3.

### Hidden Layer und Neuronen

Die Anzahl der versteckten Neuronen und *Layer* ist entscheidend für die Fähigkeit des Netzes, die zugrunde liegenden Muster in den Daten zu erfassen. Zu wenige bieten möglicherweise nicht genügend Flexibilität, während zu viele zu einer Überanpassung führen können. In der Regel wird die Anzahl der versteckten Einheiten auf der Grundlage der Anzahl der Eingaben und der Trainingsfälle gewählt und liegt oft zwischen 5 und 100. Die Regularisierung trägt dazu bei, die Überanpassung zu verringern, wenn eine größere Anzahl versteckter Einheiten verwendet wird. Die Wahl der versteckten Schichten hängt von der jeweiligen Anwendung ab und wird durch Hintergrundwissen und Experimente bestimmt.

# 4. Data

Zur Fütterung der in Kapitel 3 beschriebenen *Machine Learning Methoden* wird immer ein Datensatz mit den relevanten Merkmalen der VN benötigt. Der Aufbau, die Vorbereitung und Transformation der zu trainierenden Datenmenge sind die ersten Schritte des KDD und werden bei der technischen Umsetzung immer vor dem *Machine Learning* ausgeführt. Durch den Datencheck der Qualität und Analyse der verschiedenen Arten der Variablen kann die Qualität der danach verwendeten Modelle wesentlich verbessert werden. Dieses Kapitel dient zur Beschreibung der verschiedenen Datenquellen, den Umgang mit kategorischen und numerischen Variablen und wie die Dimension des Trainingsdatensatz eventuell reduziert werden kann.

In dieser Arbeit werden alle privaten VN der Allianz Österreich mit einem Betreuer aus dem Exklusivvertrieb (Agentur oder Außendienstmitarbeiter) innerhalb eines Jahres beobachtet. Jene VN welche in diesem Zeithorizont ein zusätzliches Risikosegment kauften werden mit 1 für das jeweilige Segment markiert, diese Variable gilt es mit Hilfe der Methoden aus Kapitel 3 für den nächsten Zeithorizont zu prognostizieren bzw. eine Wahrscheinlichkeit zu schätzen. Um auch eine verlässliche Vorhersage für Neukunden zu treffen, wird bei der Kundenart zwischen Bestands- und Neukunden unterschieden. Zudem werden nur VN berücksichtigt, welche einen aufrechten oder ruhenden Vertrag haben. Dadurch kann gewährleistet werden, dass die Kundenkontaktinformationen wie Telefonnummer oder E-Mail Adresse aktuell sind und ein Cross-Selling Angebot tatsächlich unterbreitet werden kann.

## 4.1. Datenquellen

Die Datenquellen, die für diese Arbeit genutzt werden, sind vielfältig und umfassen sowohl zum Teil öffentlich zugängliche als auch unternehmensinterne Datenbestände. Die Hauptquellen sind:

### 4.1.1. Statistische Quellen Österreichs

Ein kleiner aber wichtiger Teil der Daten stammt von statistischen Websites in Österreich, wie etwa der Statistik Austria. Diese Daten liefern wertvolle demographische und sozioökonomische Informationen für die Gesamtbevölkerung und bereichern dadurch den unternehmensinternen Datenbestand. Beispielsweise können Daten über die Bevölkerungsstand, KFZ-Zulassungen, Einkommensverteilung und regionale Unterschiede verwendet werden, um potenzielle Cross-Selling-Chancen zu identifizieren.

### 4.1.2. Unternehmensportal GFB

Ein sehr wichtiger Datenlieferant ist das Geschäftsfallbearbeitungssystem (GFB), das von Miterarbeitern der Allianz mit spezifischen Informationen über VN befüllt wird. Darunter sind detaillierte Daten zu bestehenden Kunden, Vertragsinformationen, Schadensfällen, Kontaktdaten und weiteren relevanten Merkmalen. Diese Daten sind die Grundlage für das Training der *Machine Learning Methoden*.

### 4.1.3. SAS-Datenbank

Zusätzlich zu den oben genannten Quellen werden umfangreiche Datenbestände aus der unternehmensinternen SAS-Datenbank genutzt. Diese Datenbank erweitert die Daten aus dem GFB um ein vielfaches, da dort auch historische Daten liegen. Zudem beinhaltet die SAS-Datenbank auch noch detailliertere Informationen von VNs wie Prämienbewegungen, Stornoquoten, Kundenbewertungen, Rabatte und weitere relevante Geschäftsdaten, die für die Modellierung eine Bereicherung sein können. Dadurch ist es möglich tiefergehende Einblicke in das Kundenverhalten und die bisherigen Verkaufsstrategien zu gewinnen.

### 4.1.4. Integration von Daten von Versicherungskonkurrenten

Durch den Kauf von Marktforschung und Branchenberichte oder Finanzberichte von Konkurrenten wäre es auch möglich die Datenbasis zu erweitern und mögliche Abschlüsse anderer VU vorherzusagen und diesen mit einem Angebot zuvorzukommen. Dieser Ansatz würde jedoch den Rahmen dieser Arbeit sprengen und wird deshalb nicht weiter verfolgt.

In Summe sind die verwendeten Merkmale für VN aus den drei genannten Quellen die folgenden: Bevölkerungsstand, KFZ-Zulassungen, Einkommensverteilung, Risikogruppe (Leben, Unfall, Kranken), Alter, Geschlecht, Anfangsdatum des VN, Vermittlertyp (Agent, Außendienst, Makler), Anzahl Risken, Kundenzählung, Kundenart, Haushaltsrolle, Jahresnettoprämie in Summe und aufgeteilt auf Risikogruppen, Anzahl der stornierten Verträge, Berufsgruppe, Einkommen, Anzahl der aufrechten Verträge, Anzahl der Mitglieder im Haushalt, Arbeitsverhältnis, Familienstand, Postleitzahl, Kundenbewertung von Vertragsabschluss, Kundenbewertung von Schadensabwicklung, Kundenportalaktivierung, Marketingklausel, Anzahl der Fahrzeuge, Anzahl der Wohnungen, Anzahl der Häuser, Anzahl der gemachten Offerten, Schadenhöhe, Anzahl der Schadensfälle, Anzahl anderer versicherten Risikosegmente

Datenbanken ändern sich von Jahr zu Jahr und aus Unternehmenssicht ändern sich oft auch die zu beobachtenden KPIs. *Machine Learning* erweist sich dazu als sehr nützlich, da die Modelle so gestaltet werden können, dass die Wahl der Merkmale auch im Nachhinein noch verändert werden kann. Im Grunde ist es empfehlenswert einmal jährlich einen neuen Trainingslauf durchzuführen um neue Kaufverhalten in das Modellverhalten miteinzubeziehen und falls notwendig neue Merkmale einzubauen.

## 4.2. Datenanalyse

Die Datenanalyse ist ein zentraler Schritt nach der Beschaffung des Datensatzes aus Kapitel 4.1. In diesem Kapitel werden die wesentlichen Fragen beleuchtet, die während der Datenanalyse beantwortet werden sollten, sodass vor der Implementierung der *Machine Learning Methoden* ein Verständnis der zugrundeliegenden Daten entstehen kann.

1. Datenqualität und Vollständigkeit
   - Sind alle notwendigen Daten vorhanden?
   - Gibt es fehlende Werte und wie häufig treten diese auf? Kapitel 4.3.1
   - Gibt es Ausreißer oder Fehler?
2. Art und Verteilung der Merkmale
   - Welche sind Kategorische und welche Numerische Merkmale? Kapitel 4.3.3 und 4.3.2
   - Wie hoch ist die Dimension der Kategorischen Merkmale? (z.B. Dimension 4 bei Geschlecht: M, W, D, Fehlt)
   - Wie sind die Verteilungen der einzelnen Merkmale?
3. Welche Risikosegmente sind die meistgekauften in einer Periode?

Um die oben genannten Fragen zu beantworten, ist es hilfreich verschiedene Plots und Visualisierungen zu verwenden.

1. Histogramme und Dichteplots
   - Ideal zur Untersuchung der Verteilung einzelner Merkmale
   - Beispiel: Histogramm der Altersverteilung der VN, Dichteplot der Jahresnettoprämie
2. Boxplots
   - Nützlich zur Identifizierung von Ausreißern
   - Beispiel: Boxplot der Schadenssummen pro Risikosegment
3. Scatterplot und Matrixplot
   - Diese Plots sind hilfreich um Korrelationen zwischen zwei oder mehreren Merkmalen zu visualisieren
   - Beispiel: Beziehung zwischen Einkommen und Anzahl der Risikosegmente
4. Heatmaps
   - Eine Heatmap kann verwendet werden um stark verkaufte Risikosegmente leicht zu identifizieren

## 4.3. Datenvorverarbeitung

Die Datenvorverarbeitung oder auch *Data Preprocessing* ermöglicht die Entfernung unerwünschter Daten durch Datenbereinigung, sodass der Benutzer nach dieser Phase einen Datensatz erhält, welcher sich besser für die spätere Datenmanipulation und *Learning* im KDD eignet. Die eingesetzten *Data Preprocessing Steps* können erheblich beeinflussen, welche Schlussfolgerungen aus der Analyse gezogen werden. Daher ist es notwendig, die Qualität der Daten sicherzustellen, bevor eine Analyse durchgeführt wird [45]. Wenn ein hoher Anteil an irrelevanten, redundanten oder unzuverlässigen Daten vorliegt, kann dies den Trainingsalgorithmus stark beeinträchtigen. Die Schritte der Datenvorverarbeitung nehmen gerade deshalb oft einen erheblichen Zeitaufwand im Vergleich zu der tatsächlichen Implementierung der *Machine Learning Algorithmen* ein. Beispiele für Methoden der Datenvorverarbeitung sind Datenbereinigung, Normalisierung, *One-Hot-Encoding*, Datentransformation, *Feature Selection* und *Train-Test Splitting* [45].

### 4.3.1. Fehlende und fehlerhafte Werte

Bevor man weiter in die *Data Preprocessing Steps* eintaucht, muss festgestellt werden, in welcher Form fehlende Werte ersichtlich sind. Da eine Datenmenge im Allgemeinen aus zwei verschiedenen Komponenten besteht, nämlich den Klassenzuordnungen und den Merkmalswerten [46], ergeben sich daraus wiederum zwei verschiedene Arten von fehlerhaften Daten [47]:

- *Class Noise*, entsteht bei falsch zugeordneten Klassen. Dieser Fall hat in dieser Arbeit keinen übergeordneten Stellenwert, da das Kaufverhalten durch Prämienzahlungen bestätigt werden und somit nur der Fall von fehlerhaften Eingabewerten in GFB eine Rolle spielen könnte.

- *Attribute Noise* sind z.B. fehlerhafte, fehlende oder unvollständige Merkmalswerte.

Die Vorverarbeitung von *Attribute Noise* aus der zu trainierenden Datenmenge ist in der Literatur eine empfohlene Vorgehensweise, dadurch kann eine Verbesserung der Klassifizierungsgenauigkeit erzielt werden. Anstatt Instanzen, die *Attribute Noise* enthalten, zu eliminieren, ist die Korrektur von den betroffenen Merkmalen im Allgemeinen besser [47]. Fehlerhafte Werte können z.B. durch Bereinigung von Ausreißern verbessert werden. Nach einer kurzen Analyse der Datenmenge aus dem VN-Datensatz der Allianz ist ersichtlich, dass vor allem die Anzahl der fehlenden Werte für die verschiedensten Merkmale einen großen Anteil (im vgl. zu fehlerhaften Werten) beansprucht. Merkmale mit großen Anteil fehlender Werte sind hauptsächlich Informationen, die ein VN nur freiwillig angeben oder akzeptieren muss, wie z.B. E-Mail, Handynummer oder Kundenbewertungen. Aus diesem Grund wird in dieser Arbeit mehr Fokus auf den Umgang mit fehlenden Werten gelegt. Da fehlende Werte für viele der vorgestellten Methoden in Kapitel 3 nicht verarbeitbar sind, ist die Notwendigkeit zur Lösung des Befüllens von fehlenden Merkmalen besonders hoch.

### Methoden zum Ersetzen von fehlenden Werten

Wie im obigen Absatz beschrieben, ist es wichtig fehlende Werte bzw. *Missing Values* (MV) zu ersetzen oder jene Zeilen mit MV aus der Trainingsmenge zu löschen. Ein großer Nachteil beim Löschen von MV ist, dass dabei oft ein großer Anteil der Datenmenge wegfällt und dabei auch Muster verloren gehen, welche einen positiven Informationsgewinn für ein Modell hätten [45]. Ein intuitiver Ansatz zum Ersetzen wäre z.B. alle fehlenden numerischen Merkmale mit dem Mittelwert oder Median aus dem Trainingsdatensatz zu befüllen, beim Mittelwertersatz würde dabei der Gesamt-Mittelwert unverändert bleiben aber dafür die Gesamt-Varianz unterschätzt. Ein einfaches Ersetzen der Werte durch neue, statistisch sinnvolle (Mittelwert, Median, Mode), ohne zu vermerken, dass es sich hierbei um MV handelt, führt zu einem Informationsverlust der Datenmenge. Vor allem bei einer größeren Menge an fehlenden Daten, kann es vorkommen, dass das Muster davon die wichtigste Information zur Modellierung ist [45]. Somit sollte eine geeignete Methode gewählt werden, welche zuerst das Muster der MV erkennt und anschließend die Werte mit dem 'bestmöglichen' Schätzer ersetzen. Ein solcher Schätzer sollte die wichtigsten Eigenschaften der Datenmenge so wenig verändern wie möglich. Einige dieser Eigenschaften sind der bereits erwähnte Mittelwert, die Varianz oder auch Abhängigkeiten der Merkmale untereinander. Eine einfache Methode zum Ersetzen von MV wäre eine Lineare Regression, also eine lineare Funktion in Abhängigkeit der anderen Merkmale. Eine andere Methode fehlende Werte aufzufüllen ist ein iterativer Lernprozess, welcher einen Teil der MV befüllt und in bekannte Werte umgewandelt, die dann für den nächsten Schritt der Imputation fehlender Werte verwendet werden. Diese Art von *Iterative Imputation* für MV hat in den letzten Jahren stark an Aufmerksamkeit gewonnen. Laut einer Metastudie aus dem Jahr 2022 [48] werden immer häufiger *Machine Learning Methoden* als *Iterative Imputation* für MV verwendet. Während bei sehr großen Datenmengen Methoden wie *Deep Learning* besser abschneiden, erzielen *K-Nearest-Neighbors* und *Support Vector Machines* bei kleineren Datenmengen eine höhere Genauigkeit.

### K-Nearest-Neighbors Imputationsmethode

Die Methode des *K-Nearest-Neighbors Imputer* zur Schätzung von MV berechnet den Wert eines bestimmten fehlenden Datenpunktes anhand der Werte seiner Nachbarn, wobei die Nähe der Nachbarn in einem $n$-Dimensionalen Raum anhand der Euklidischen Metrik gemessen wird. Damit auch ein Abstand für fehlende Daten berechnet werden kann wird eine Gewicht $w_i$ in die Metrik hinzugefügt, welcher den Wert 0 hat falls ein Datenpunkt fehlt und andernfalls gleich 1 ist [49]:

$$d_j = d(F_i, F_j) = \sqrt{\sum_{k=1}^{n} w_i \cdot (F_{i,k} - F_{j,k})^2}$$

Die tatsächliche *Imputation* für einen fehlenden Wert $MV_i$ kann anschließend mit einem Mittelwert der nächsten Nachbarn berechnet werden:

$$MV_i = \frac{1}{\sum_{j=1}^{k} w(d_j)} \sum_{j=1}^{k} w(d_j) \cdot F_j$$

Bei Sonderfällen wie, falls die Anzahl der $n$ Nachbarn nicht existiert, wird standardmäßig der Mittelwert der Trainingsmenge verwendet. Auch bei dem Fall, dass ein Merkmal überall fehlt, dann wird dieses aus dem Datensatz komplett entfernt. Wird die k-NN Methode nun iterativ angewandt bis die *imputed values* zu einer a-priori definierten Grenze konvergieren, dann spricht man von einem *K-Nearest-Neighbors Iterative Imputer*. Das Iterieren einer solchen Methode ist durchaus rechenintensiv und ist deshalb bei sehr großen Datenmengen nicht praktikabel, eine einfache k-NN Methode führt dabei auch schon zu guten Ergebnissen [50].

### 4.3.2. Numerische Merkmale

Da Merkmale wie z.B. Schadenhöhe, Jahresnettoprämie und Kundenbewertungen eine sehr unterschiedliche Struktur aufweisen, liegt es Nahe, dass ein Teil der Datenvorverarbeitung sich damit beschäftigt die numerischen Merkmale auf die gleiche Struktur zu bringen.

#### Normalisierung

Die Normalisierung von Merkmalen ist ein Prozess, bei dem die Werte auf eine gemeinsame Skala normiert werden. Damit wird verhindert, dass kein Merkmal aufgrund seiner Skala andere Merkmale dominiert. Zwei Normalisierungstechniken, welche häufig benutzt werden, sind:

- Min-Max Skalierung: Hier werden die Werte auf einen festen Bereich ($[0, 1]$ oder $[-1, 1]$) skaliert.

$$F_{\text{MinMax}} = \frac{F - F_{\min}}{F_{\max} - F_{\min}}$$

- Z-Score Standardisierung: Hier werden die Werte auf einen Mittelwert von 0 und Standardabweichung von 1 transformiert.

$$F_{\text{standard}} = \frac{F - \mu}{\sigma}$$

#### Verteilungs-Transformation

Da einige der Algorithmen davon ausgehen, dass die Eingabedaten einer Normalverteilung folgen, ist es ratsam die numerischen Werte mithilfe von Transformationsverfahren im Vorhinein anzupassen. Einige bekannte Transformationen dafür sind:

- Box-Cox Transformation [51]: Eine Transformation für positive Werte.

$$F' = \begin{cases} \frac{F^\lambda - 1}{\lambda} & \text{wenn } \lambda \neq 0 \\ \log(F) & \text{wenn } \lambda = 0 \end{cases}$$

- Yeo-Johnson Transformation [52]: Eine Erweiterung der Box-Cox Transformation für

unbegrenzte und negative Werte.

$$F' = \begin{cases} \frac{(F+1^{\lambda})-1}{\lambda} & \text{wenn } F \geqslant 0 \text{ und } \lambda \neq 0 \\ \log((F+1)) & \text{wenn } F \geqslant 0 \text{ und } \lambda = 0 - \frac{(-F+1)^{2-\lambda}-1}{2-\lambda} \\ \text{wenn } F < 0 \text{ und } \lambda \neq 2 & \\ -\log((-F+1)) & \text{wenn } F < 0 \text{ und } \lambda = 2 \end{cases}$$

- Quantil Transformation: Hier werden die Quantile der Daten auf die Quantile einer Normalverteilung abgebildet.

### 4.3.3. Kategorische Merkmale

Die Vorverarbeitung von kategorischen Merkmalen erfordert aufgrund ihrer Struktur (z.B. Buchstaben anstatt Zahlen) andere Techniken als wie bei numerischen Merkmalen. Zudem können kategorische Merkmale auch geordnet sein, müssen aber nicht (z.B. Alter oder Geschlecht). Im Gegensatz zu den numerischen Merkmalen können kategorische Merkmale nicht direkt in das Modell eingegeben werden. Deshalb wird in der Datenvorverarbeitungsphase die verschiedenen Merkmale gekennzeichnet, ob das Merkmal *nominal* (ungeordnete) oder *ordinal* (geordnet) ist.

#### One-Hot Kodierung

*One-Hot Encoding* oder auch 1-aus-n Code ist eine Technik, welche sich eignet um nominale Merkmale in mehrere Spalten mit Werten $\{0, 1\}$ zu transformieren. Wie sehr leicht erkennbar ist, kann die Dimensionalität bei Merkmalen mit vielen Kategorien dadurch sehr schnell explodieren.

#### Ordinale Kodierung

*Ordinal Encoding* ist eine Kodierung für kategorische Merkmale welche die Information der Reihenfolge auch nach Kodierung beibehält. Diese Technik ordnet aufgrund der Reihenfolge ganzzahlige Werte zu.

#### Imputation

Fehlende Werte bei kategorischen Merkmalen werden meistens durch die häufigste Kategorie oder eine neue Kategorie (z.b. 'unbekannt') ersetzt. Die in Kapitel 4.3.1 beschriebenen Methoden wie *Iterative Imputer* benötigen immer numerische Eingabewerte und da dies nicht die Natur der kategorischen Merkmalen entspricht, werden meistens einfachere Methoden genutzt.

### 4.3.4. Merkmalsauswahl

Da durch die Verarbeitung von kategorischen Merkmalen, insbesondere durch *One-Hot Encoding*, die Dimension der Merkmale sehr schnell wächst, stellt sich die Frage, ob es nicht auch Methoden gibt um die Merkmale selektiv zu verwerfen und nur den Teil mit

der höchsten Information zu behalten. Tatsächlich ist dieser *Feature Selection* Schritt ein wichtiger Bestandteil in der Datenvorverarbeitung. Durch eine geringere Anzahl von Merkmalen kann ein Modell schneller trainiert werden und weißt oft auch bessere Performance und ein geringeres *Overfitting* vor. Da es ein großes Spektrum an *Feature Selection Methods* gibt, sind hier ein paar Beispiele nach den verschiedenen Methoden:

- Filter Methoden: Statistische Maße werden verwendet um die Relevanz eines Features zu berechnen. Korrelationskoeffizienten, Chi-Quadrat Test und ANOVA (*Analysis of Variance*) sind einige Beispiele.
- Wrapper Methoden: Diese Methoden verwenden einen iterativen Algorithmus welcher aus verschiedenen Untermengen der Merkmale diejenigen auswählt welche die beste Performance liefern. Beispiele sind *Forward/Backward Selection* und *Recursive Feature Elimination.*
- Embedded Methoden: Diese haben vor allem bei Neuronalen Netzen einen großen Nutzen und werden häufig eingesetzt. *Embedded* bedeutet in diesem Sinne, dass die *Feature Selection* während des Trainings passiert und nicht als Vorverarbeitungsschritt. Beispiele sind die LASSO Methoden (*Least Absolute Shrinkage and Selection Operator*), welche eine $L1$ oder $L2$ Norm zur Bestrafung des Lernprozesses hinzufügen. Dadurch konvergieren die Gewichte für nicht aussagekräftige Merkmale Richtung 0 und werden somit aus dem Lernprozess ausgeschlossen.

### Principal Component Analysis

Die Hauptkomponentenanalyse oder meistens auch *Principal Component Analysis* (PCA) ist eine *Unsupervised Machine Learning Methode*, welche häufig dazu genutzt wird, um die Dimension der Eingabeparameter zu verringern, siehe Abbildung 4.1. Damit eignet sich PCA auch als *Feature Selection Method* und soll im folgenden näher erläutert werden. PCA wandelt die ursprünglichen Variablen in sogenannte *Principal Components* um. Mit diesen kann dann in weiterer Folge gearbeitet und die relevanten Merkmale ausgewählt werden. Die erste Komponente ist eine Linearkombination der ursprünglichen Variablen, welche die Varianz im Datensatz maximiert. Die zweite Komponente erklärt dann die maximale Varianz der verbleibenden Varianz ohne der ersten Komponente. Dieser Prozess wird fortgesetzt, bis die gesamte Varianz des Datensatzes erreicht ist. Voraussetzung dafür sind, dass die Komponenten orthogonal zueinander sind. Für eine detaillierte Beschreibung der PCA und der Herleitung dessen, sei auf die Literatur verwiesen [53] [54]. Mithilfe der Kovarianzmethode können die *Principal Components* durch Eigenwertzerlegung der Stichproben-Kovarianz berechnet werden, die Komponenten sind im Grunde die Eigenwerte der Kovarianzmatrix [55] . Nach Bestimmung der Komponenten werden jene verworfen, welche nur einen geringen Beitrag zur Gesamtvarianz aufweisen. In der Praxis werden dabei die Eigenwerte der Größe sortiert und deren Varianz kumuliert und ab einer vorgegebenen Grenze werden alle übrigen Eigenwerte verworfen. Anschließend wird die ausgewählte Teilmenge der Eigenwerte mit dem Anfangsdatensatz multipliziert und somit eine neue Basis mit geringerer Dimension hergeleitet. Die PCA kann die Dimension von den Daten zum Training von *Machine Learning Methoden* erheblich reduzieren, was sie zu einem leistungsstarken

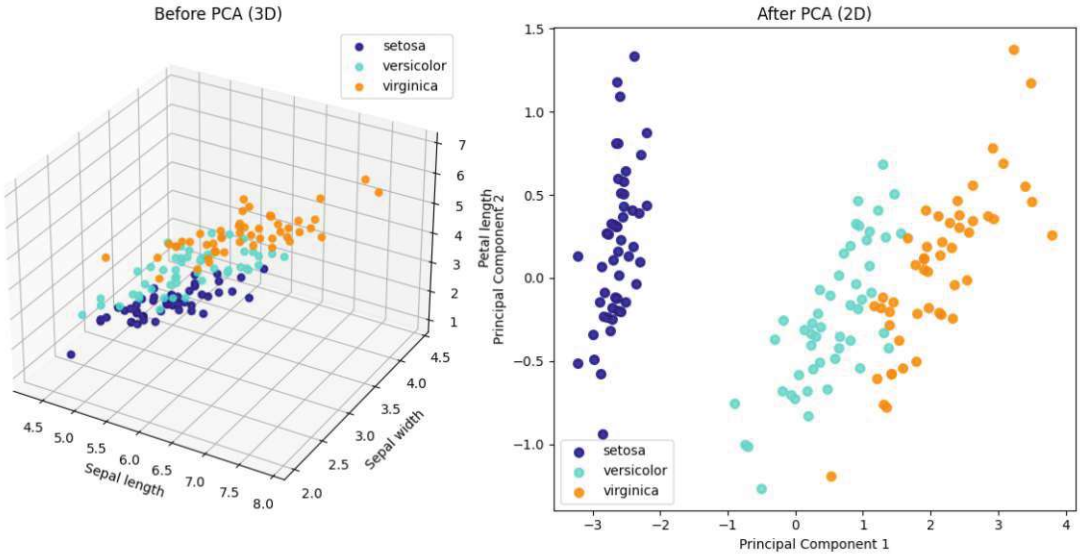


Abbildung 4.1.: PCA Methode angewandt auf den Iris-Testdatensatz mit drei Eingabeparameter und nach PCA auf zwei Komponenten reduziert

Werkzeug in der *Data Preprocessing Phase* macht. An dieser Stelle sei darauf hingewiesen, dass bei Anwendung von PCA und dem Verwurf von Merkmalen mit geringer Varianz, die Leistung von Klassifizierungsmodellen auch negativ beeinflusst werden kann [56] .

### 4.3.5. Klassen Ungleichgewicht

Die Verteilung der Klassen in positive und negative wirkt sich sehr stark auf das Ergebnis des Klassifizierungs-Algorithmus aus. Meistens ist die 'interessantere' Klasse jene, welche in der Minderheit ist. Bereits in den 1990er Jahren wurde man auf das Problem des Klassen Ungleichgewichts (*Class Imbalance*) aufmerksam [57] [58]. Um das Ungleichgewicht zu verringern, wurden verschiedene *Oversampling* und *Undersampling* Techniken eingesetzt und verglichen. Methoden wie das *Random Oversampling* duplizieren die Datenreihe der Minderheitsklasse bis eine gewünschte ähnliche Verteilung zwischen positiven und negativen Klassen erreicht ist. Dadurch wird aber auch das Risiko zum *Overfitting* erhöht, da der Algorithmus zum Klassifizieren lernt die duplizierten Stichproben vorherzusagen. Außerdem wir die Trainingsmenge erhöht, was unmittelbar zu höheren Rechenkosten und längeren Trainingszeiten führt. Dies führt auch zur Überlegung, was wäre, wenn die Anzahl der Mehrheitsklasse reduziert wird, um ein Gleichgewicht zu erlangen. Der *Undersampling* Ansatz kann jedoch zu einem Verlust von wichtigen Informationen führen (ähnlich wie beim Verwerfen von *Missing Values*).

Um die Nachteile der herkömmlichen *Oversampling* und *Undersampling* Methoden zu überwinden, entwickelten Chawla, Bowyer et. al. [59] die *Synthetic Minority Oversampling Technique* (SMOTE). SMOTE erzeugt mittels Interpolation zwischen den Minderheitsklas-

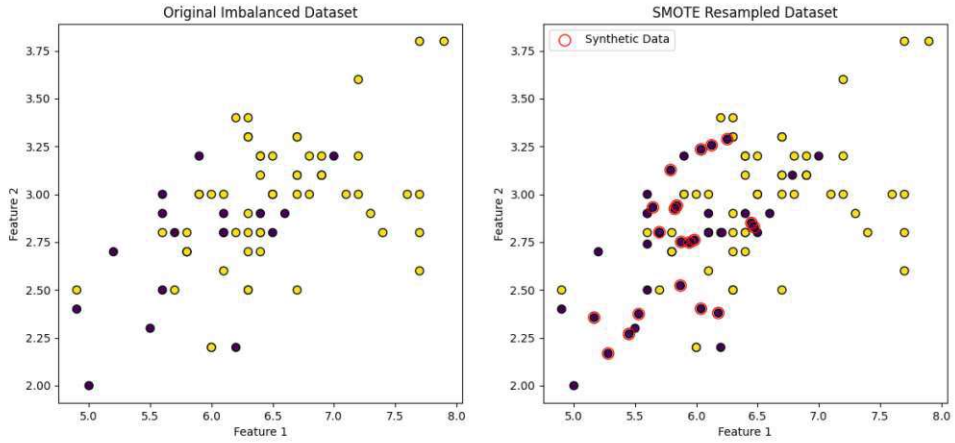


Abbildung 4.2.: SMOTE Methode angewandt auf eine Datenmenge mit *Class Imbalance*

sen neue synthetische Datenreihen. Dieses Verfahren erzeugt nicht nur ein Klassen Gleichgewicht, sondern stellt auch sicher, dass die synthetischen Stichproben verallgemeinert sind und damit wird das Risiko zum *Overfitting* reduziert. Die Vorgehensweise zur Berechnung des SMOTE kann wie folgt vereinfacht beschrieben werden:

1. Auswahl der Minderheitsproben: Für jeden Punkt in der Minderheitsklasse werden die $k$ nächsten Nachbarn (k-NN) gekennzeichnet.
2. Erzeugung von neuen synthetischen Datenpunkten:
   - Berechnung der Differenz zwischen dem Datenpunkt aus 1 und seinen Nachbarn.
   - Multiplizieren dieser Differenz mit einer Zufallszahl zwischen 0 und 1.
   - Das Ergebnis wird zum ursprünglichen Merkmalsvektor der Minderheitsklasse hinzugefügt. Dadurch entsteht eine neue synthetische Stichprobe entlang einer Linie zwischen der ausgewählten Minderheit und ihren Nachbarn.

Die Abbildung 4.2 zeigt wie ein Datensatz mit *Class Imbalance* durch Anwendung von SMOTE zu einem ausgewogeneren Datensatz führt. Es sei angemerkt, dass durch die stochastische Natur von SMOTE jede Anwendung andere synthetische Stichproben erzeugt. SMOTE wurde mittlerweile bereits in vielen verschiedenen Bereichen erfolgreich eingesetzt und gilt als State of the Art bei *Class Imbalance* [60].

### 4.3.6. Trainings-, Validierungs- und Testdaten

Bei *Machine Learning* geht es darum, Vorhersagen aus der Datengrundmenge zu lernen. Diese Algorithmen werden zunächst auf einem Trainingsdatensatz trainiert. Um in weiterer Folge die *Hyperparameter* der Algorithmen zu verändern und das Modell zu verbessern (*tuning*), wird ein Validierungsdatensatz verwendet. Die *Performance* auf den Validierungsdaten ist ausschlaggebend für das *Tuning*. Zuallerletzt wird dann das Trainierte

und verbesserte Modell einem, für das Modell, unbekannten Datensatz ausgesetzt. Dieser als Testdatensatz bekannte Teil der Datengrundmenge dient dazu die *Performance* des fertigen Modells zu messen. Mittels den 'ungesehenen' Daten aus dem Testdatensatz kann eine Erkenntnis abgeleitet werden, ob das Modell nur auf die trainierte Menge funktioniert oder auch neue Daten gut vorhersagen kann. Zusammengefasst dient eine Aufteilung der Grunddaten zum Trainieren, Verbessern und zur Untersuchung zwecks *Overfitting*. Es gibt keine vorgegebene Ideale Unterteilung, in wieviel Prozent in jedem Split sein sollen. Häufig gewählte Splits sind 80/10/10, 70/15/15 oder 60/20/20 [61]. Die Wahl der Aufteilung hängt oft von der vorgegebenen Rechenkapazität oder auch der gewünschten Kalibrierung ab.

# 5. Implementation

In diesem Kapitel werden die praktischen Aspekte und Resultate der Entwicklung eines *Recommender System* zum *Cross Selling* von Versicherungsprodukten unter Einsatz von maschinellem Lernen behandelt. Die theoretischen Grundlagen, die in den vorherigen Kapiteln besprochen wurden, dienen als Basis für diese Implementierung.

Die Implementierung wurde auf einer von der Allianz bereitgestellten Data Platform durchgeführt, wobei das Programm in einem Jupyter-Notebook geschrieben wurde und zahlreiche verschiedene Bibliotheken verwendet wurden. Der Großteil der Algorithmen wurde mit Hilfe der Bibliothek *PyCaret* durchgeführt, welche eine Zusammenfassung von mehreren ML-Algorithmen für Klassifikations- und Regressionsprobleme ermöglicht. In Kombination mit den Bibliotheken *xgboost*, *lightgbm*, *catboost*, *TensorFlow*, *sci-kit learn* und *Keras* konnten alle in dieser Arbeit erwähnten Methoden implementiert werden. Während *imbalanced-learn* zur Lösung von Klassenungleichgewichten beitrug. Als Deep-Learning-Modelle kamen *TensorFlow* und *Keras* zum Einsatz.

Das Ziel dieser Implementierung war es, ein robustes System zu entwickeln, das in der Lage ist, den Abschluss von Versicherungen vorherzusagen und damit die Erstellung von personalisierten Empfehlungen für VN zu unterstützen. Der erste Schritt bestand darin, die Daten für die Analyse vorzubereiten und aus den verschiedenen Versicherungssparten gruppierte Risikoprofile zu erstellen, welche die gängigsten Versicherungsprodukte auf einer Risikoebene zusammenführen (Kapitel 5.1).

Nach der Vorverarbeitung der Daten wurden in der nächsten Phase die zehn verschiedenen Modelle (Naive Bayes, Logistische Regression, LDA, QDA, Decision Trees, Random Forest, XGBoost, LightGBM, Catboost und Neuronale Netzwerke) trainiert und dessen Lernprozess analysiert (Kapitel 5.2). Die Modelle wurden schlussendlich anhand ihrer Leistungsmetriken wie *Precision*, *Recall*, und *F1-Score* bewertet, um das beste Modell auswählen zu können (Kapitel 6).

## 5.1. Vorverarbeitung

An dieser Stelle sei noch einmal explizit darauf hingewiesen, dass für die für diese Arbeit verwendeten versicherungsspezifischen Kundendaten von der Allianz bereit gestellt wurden und ohne diese Zusammenarbeit die Entwicklung dieses *Cross-Selling Recommender Systems* nicht möglich gewesen wäre.

Der erste Schritt, bevor die eigentliche Implementation in Python überhaupt stattfinden kann, ist es die ausgewählten Kundenmerkmale aus der Datenbank herauszuziehen, auf VN Ebene zu aggregieren und für die möglichen Cross-Selling Produkte ein Risikoprofil zu generieren. Diese Risikoprofile sind eine Zusammensetzung vieler einzelner Risken. Dadurch können auch ältere Produktgenerationen mit den aktuellen und zukünftigen Produkten

| KFZ Privat | Sach Privat | Gesundheit | Unfall |
|---|---|---|---|
| 47, 43% | 74, 47% | 95, 41% | 95, 97% |

Tabelle 5.1.: Verkaufspotential für die gegebenen Risikoprofile als Anteil aller VN der Allianz

verglichen werden. Die Zusammensetzung dieser Risikoprofile sei wie folgt gegeben (an dieser Stelle werden nur die für diese Arbeit relevanten Profile genannt):

1. KFZ Privat Profil:
   - KFZ-Haftpflicht
   - KFZ-Nebenrisken (Voll- und Teilkasko, Insassen-Unfall, KFZ-Service)
2. Sach Privat Profil:
   - Haushalt/Eigenheim (u.a. Glasbruch, Elektrogeräte, Feuer-Zivil, Sturmschäden, Einbruchdiebstahl)
   - Haftpflicht (u.a. Zivilhaftpflicht, Vermögenshaftpflicht)
   - Rechtsschutz
3. Gesundheit Profil:
   - Krankengeld
   - Gesundheitsvorsorge
   - Krankenhauskosten
   - Auslandsurlaub
   - Ambulanztarif
4. Unfall Profil:
   - Einzel-, Kollektiv- und Familienunfall
   - Arbeitslosenzusatzversicherung

Durch diese Definition kann nun das mögliche Cross-Selling Potential (Anteil der VN mit einem Vertrag in diesen Risikoprofil) gemessen werden. Das Ergebnis ist hier in Tabelle 5.1 in Prozent und gleicher Reihenfolge wie bei den Profilen dargestellt.

Ein Verkaufspotential von über 95% bedeutet, dass nur eine geringe Menge an VN ein solches Produkt besitzen und dadurch wahrscheinlich auch nur sehr wenige solcher Produkte in einem Jahr gekauft werden. Tatsächlich zeigt 5.2, dass dies auch der Fall war und dadurch ein Lernprozess über ML im Vergleich zu den höheren Verkaufszahlen, wie etwa bei KFZ und Sach, sich als schwierig erweisen wird. Da das Kaufverhalten von Gesundheit und Unfall sich sehr ähnlich verhält, wird in weiterer Folge das Risikoprofil Unfall verworfen und nicht weiter als Zielvariable betrachtet. Da die Anzahl der gekauften Produkte im Ungleichgewicht mit der Gesamtanzahl ist, ist an dieser Stelle schon ersichtlich, dass ein *Resampling* der Daten notwendig sein wird.

| KFZ Privat | Sach Privat | Gesundheit | Unfall |
|---|---|---|---|
| 30, 44% | 10, 41% | 0, 92% | 0, 92% |

Tabelle 5.2.: Tatsächlich gekaufte Produkte aus dem entsprechenden Risikoprofil als Anteil aller VN der Allianz

Bevor das Klassen-Ungleichgewicht in der Datenmenge verändert werden kann, sind noch weitere Analysen der Daten notwendig. Um die für die Analysen benötigten Grafiken erstellen zu können, muss untersucht werden, wieviele Datenpunkte fehlend sind und was dies in weiterer Folge bedeuten kann. Zunächst wurde ein Prozentsatz für die fehlenden Punkte pro Merkmal berechnet. Da viele der numerischen Merkmale aus den bezahlten Prämien pro Sparte, der ausbezahlten Schäden und Vertragsanzahl bestehen und diese im System korrekt vermerkt werden müssen, ist der Anteil der fehlenden Werte bei fast allen unter 2% und nie über 10%. Diese wurden mit der iterativen k-NN Methode imputiert. Es wurden auch andere Imputationsmethoden getestet. Da jedoch der Anteil der fehlenden Werte sehr gering ist, konnte kein signifikanter Unterschied in der Leistung der Klassifizierungsmethoden festgestellt werden. Es sei noch darauf hingewiesen, dass Schäden nicht gemeldet werden und somit der Datenstand in der Datenbank nicht der Realität entspricht. Der Fokus dieser Arbeit liegt nichtsdestotrotz darin, das beste Modell zu finden und nicht auf der Analyse und Bewertung von möglichen *Moral Hazards*. Im Weiteren wurden die kategorischen Merkmale nach fehlenden Werten untersucht und es stellt sich heraus, dass sich hier die größten Lücken befinden. Einer der Hauptgründe für große, teils bis zu 70% fehlenden Datenpunkten, ist jener, dass bei Vertragsabschluss viele Merkmale wie Beruf, Risikogruppe oder Einkommen nicht abgefragt werden bzw. freiwillig befüllbar sind. Manche andere Merkmale wie Angabe der E-Mail Adresse und Handynummer oder Bestätigung zur Benutzung von Kundenportal, Marketingklausel oder elektronische Polizze werden explizit von der Allianz angefragt und können von Seiten des VN abgelehnt oder ignoriert werden. Aus diesem Grund wurden alle Merkmale mit einem Anteil fehlender Werte über 10% verworfen und nicht weiter verwendet. Die restlichen fehlenden Datenpunkte wurden mit der MODE Methode aufgefüllt.

Als nächster Analyseschritt der Datenmenge wurden Histogramme für alle kategorischen Merkmale erstellt. Bei einigen von diesen Merkmalen ist erkennbar, dass diese aus sehr vielen verschiedenen Kategorien bestehen. So besteht z.B. das Alter als Merkmal aus mindestens 100 verschiedenen Kategorien. Da die kategorischen Variablen im Nachfolgenden durch *One-Hot-Encoding* aufgeteilt werden, würden aus dem einen Merkmal Alter 100 Merkmale Alter. Hierbei ist es ratsam, die Dimensionen zu verringern indem Gruppen aus diesen Merkmalen gebildet werden (siehe 5.1 und 5.2). Dieses Verfahren wurde auch bei Merkmalen wie Kundengruppe, Postleitzahl und Vermittlertyp durchgeführt (siehe die Verteilung der Kundengruppe stellvertretend für alle diese Verfahren 5.3).

Nach der Analyse des Boxplots der verschiedenen Eingabevariablen wird deutlich, dass diese eine starke Links-Schiefverteilung aufweisen (siehe 5.4 und 5.5). Dadurch kann bei hochdimensionalen Merkmalen, eine Stufe gegenüber anderen Stufen dominieren. Dies bedeutet, dass die Informationen, die ein solches Merkmal liefert, nicht sehr unterschiedlich sind. Für ein ML-Modell ist ein solches Merkmal möglicherweise nicht sehr aussagekräftig

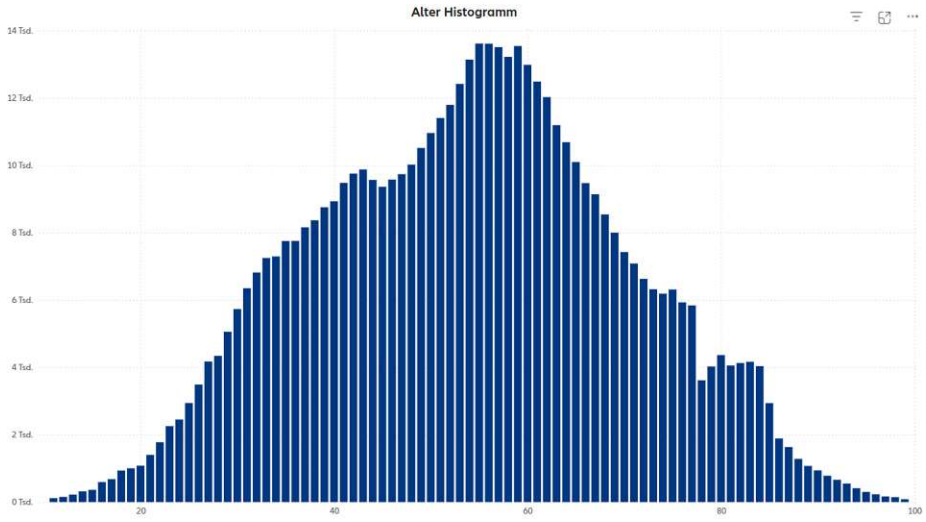


Abbildung 5.1.: Histogramm von Alter

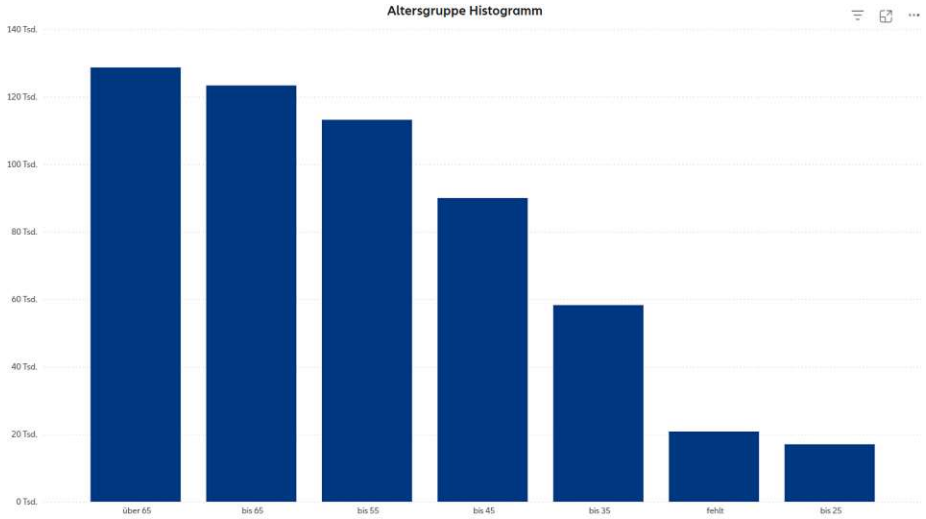


Abbildung 5.2.: Histogramm von Altersgruppe

und kann daher bei der Modellierung ignoriert werden. Die Grenze für die niedrige Varianz wurde im Preprocessing-Schritt auf den Standardwert 0,1 gesetzt. Auch eine hohe Multikollinearität zwischen zwei Merkmalen kann sich negativ auf lineare Modelle auswirken. Es wurde bei allen Merkmalen welche einen Korrelationskoeffizienten zu einem anderen Merkmal über 0,95 haben, eines der beiden weggelassen. Darüber hinaus kann eine solche schiefe Verteilung problematisch sein, da einige der benutzten Methoden davon ausgehen, dass die Daten normalverteilt sind oder zumindest symmetrisch um ihren Mittelwert verteilt (vgl. Kapitel 4.3.2). Weiters haben auch die Größe einer Variable einen direkten Einfluss auf das Modell. Aus diesem Grund ist eine Transformation auf eine Gaußsche Verteilung und eine Normalisierung der Daten auf das Intervall [0,1] notwendig. Um dies zu erzielen wur-

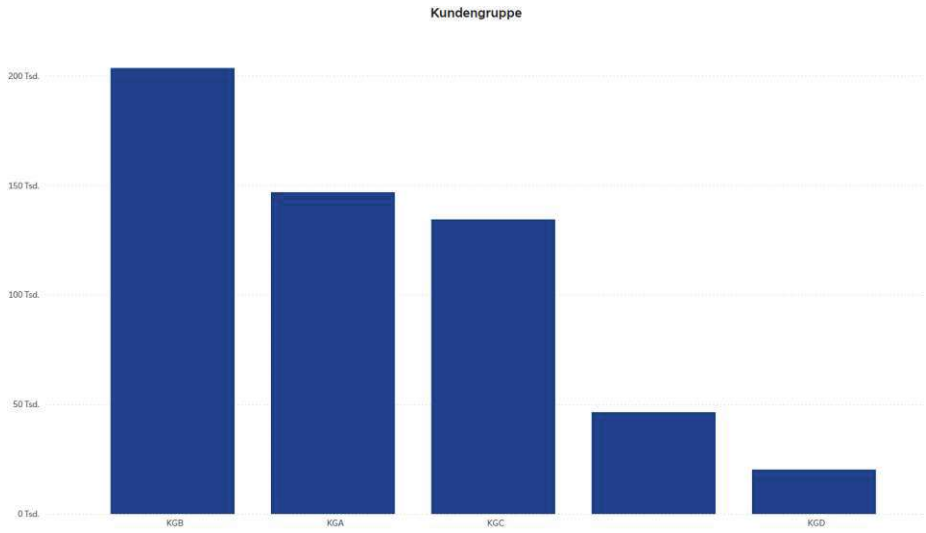


Abbildung 5.3.: Histogramm von Kundengruppe

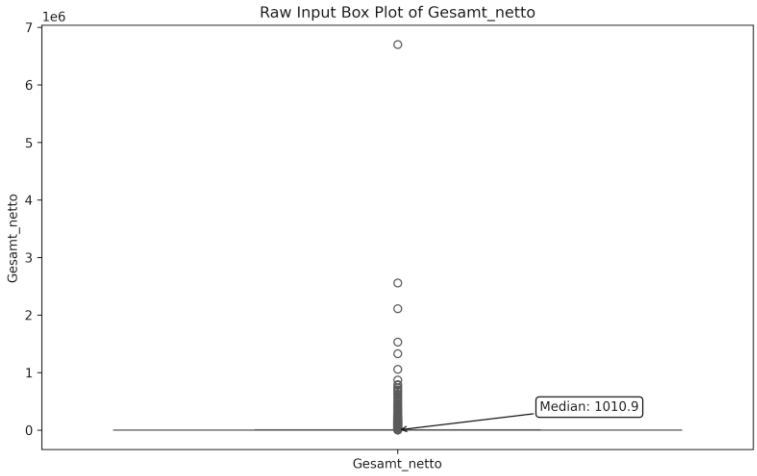


Abbildung 5.4.: Box Plot von unverarbeiteter Gesamt Nettoprämie

den die numerischen Merkmale zuerst auf dem Einheitsintervall skaliert und im nächsten Schritt einer Transformation unterzogen. Die Bilder 5.6 bis 5.9 zeigen diese Schritte anhand der Box Plots von dem gegebenen numerischen Merkmal. Die Transformation wurde mit der *PyCaret* Bibliothek durchgeführt, wobei dort standardmäßig die Yeo-Johnson Methode ausgewählt ist. Auf die Quantil Methode wurde bewusst verzichtet, da dadurch lineare Abhängigkeiten verloren gehen können.

Nachdem alle signifikanten Merkmale ausgewählt, normalisiert und transformiert wurden, wurden die kategorischen Merkmale noch dem *One-Hot Encoding* unterzogen. Damit

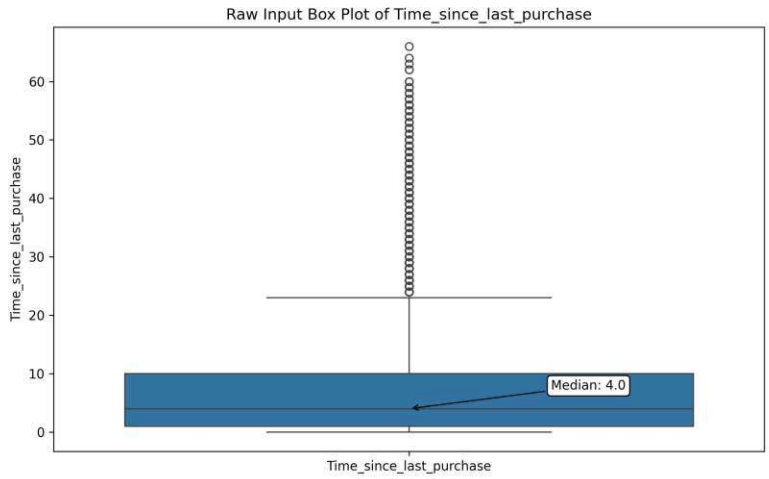


Abbildung 5.5.: Box Plot von unverarbeitetem Merkmal für letzten Kauf

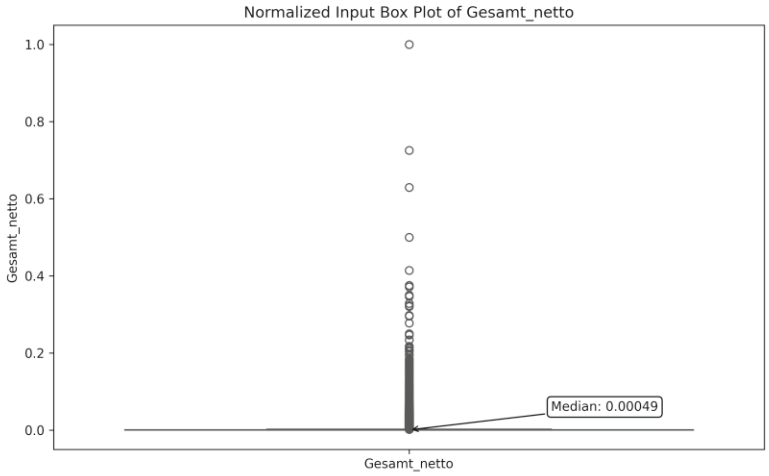


Abbildung 5.6.: Box Plot von normalisierten Gesamt Nettoprämie

fehlt es nur noch das Klassenungleichgewicht zu bereinigen. Hierfür wurde die state of the art SMOTE-Methode benutzt. Nachdem der Datensatz final aufbereitet worden ist, konnte ein Train-Test-Split von 70/30 angewendet werden. Der Train-Test-Split wählt immer per Zufallsprinzip 70% der SMOTE-Datenmenge als Trainingsmenge aus und der Rest wird als Testdatensatz verwendet. Da die Menge der gekauften Produkte für Gesundheit immer noch sehr gering sind, wurde ein höherer Test-Split gewählt. Dadurch soll ein Träger für die „gekauft “= 1 garantiert werden und die Resultate damit auch stabilisiert werden. Ein weiterer Faktor für die Stabilität eines Modells ist die *k-fold Cross Validation*, diese wurde

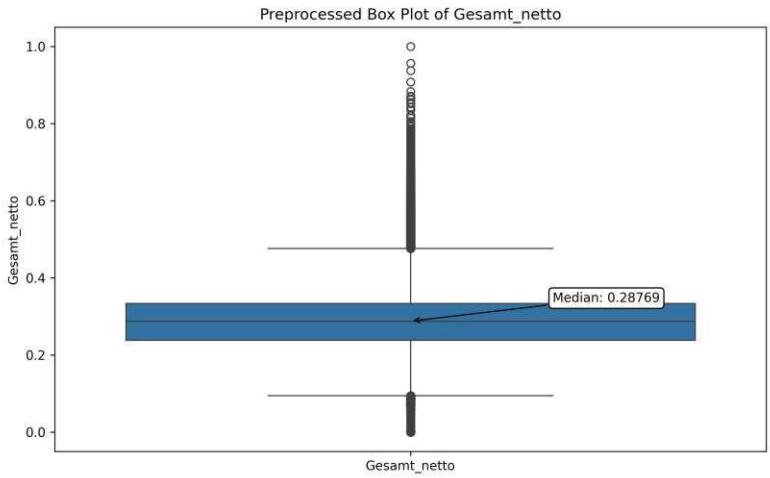


Abbildung 5.7.: Box Plot von transformierter und normalisierter Gesamt Nettoprämie

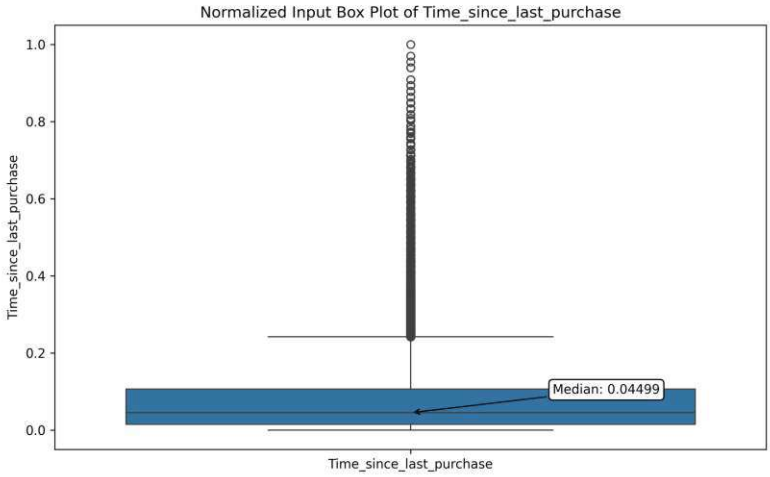


Abbildung 5.8.: Box Plot von normalisiertem Merkmal für letzten Kauf

bei allen Trainingsläufen auf 5 gesetzt (ein höherer Wert würde eine längere Trainingszeit mit sich bringen und dies war auf Grund des von der Allianz bereitgestellten Server nicht möglich). Der finale vorverarbeitete Datensatz hat nun aus den ausgewählten signifikanten 60 Merkmalen, 141 erzeugt und zugleich die Datenlänge um bis zu 40% verlängert (unterschiedlich für KFZ, Sach und Gesundheit). Der Grund dafür ist zum einen das *One-Hot Encoding* und zum anderen die SMOTE-Methode, welche neue synthetische Datenpunkte erzeugt. Bei einer so großen Matrix stellt sich die Frage, ob es mit Dimensionsreduktionen möglich ist, bessere Resultate zu erzielen. Wie sich herausstellte, kann die Verwendung von

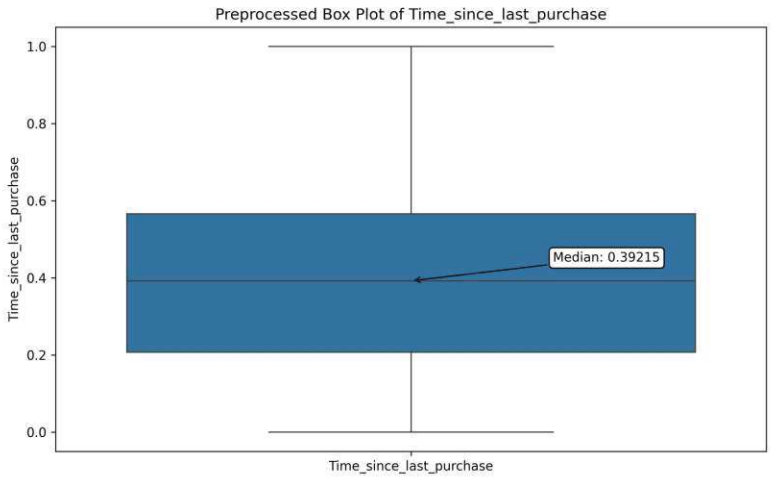


Abbildung 5.9.: Box Plot von transformiertem und normalisiertem Merkmal für letzten Kauf

Dimensionsreduktionen mit PCA und *Select k-Best* einen großen Unterschied an der Modelleistung mit sich bringen [62]. Daher werden die Resultate für jede ausgewählte Dimensionsreduktionsmethode in Kapitel 6 gegenüber gestellt. Ausgewählt wurden zum einen der Grunddatensatz mit den 141 transformierten Merkmalen, des Weiteren eine PCA-Methode und noch weitere drei Datenmengen, welche einem *Select k-Best* unterzogen wurden (jeweils $\frac{3}{4}, \frac{1}{2}, \frac{1}{4}$ des vorverarbeiteten Grunddatensatzes). Eine *Sequential Feature Selection Methode* wurde nicht verwendet, da auch hier der Rechenaufwand zum Vorbereiten des Datensatzes zu hoch für die Allianz Datenplattform war.

## 5.2. Modell Analyse

Nach der Analyse der Grunddaten und der gewünschten Vorverarbeitung kann der Trainingsdatensatz nun in die einzelnen Modelle aus Kapitel 3 eingespeist werden. Die Modelle werden im Folgenden immer mit deren Anfangsbuchstaben o.ä. abgekürzt (siehe Abkürzungsverzeichnis).

Um bei den einzelnen Modellen auch tatsächlich das Bestmöglichste auswählen zu können, wurden für alle, bis auf NB und LR (diese dienen lediglich als Vergleichsmodell zu den eigentlichen ML Methoden), verschiedene Hyperparameter verwendet. So wurde z.B. bei den Baummodellen (DT, RF, XGB, LGBM und CatB) jeweils verschiedene Tiefen der zugrundeliegenden Baumstruktur trainiert und danach das Beste ausgewählt. Bei dem LDA wurde zwischen einer *Singular value decomposition* und einem *Least squares solution* unterschieden. Bei der QDA wurden verschiedene Regulationsparameter miteinander verglichen. Für die NN wurde ein sogenannter *grid search* verwendet um aus einer Reihe von Neuronen- und Layeranzahl sowie verschiedener Aktivierungsfunktionen und Batchsize, das beste Modell auszuwählen. Auch hier sei wieder vermerkt, dass durch eine begrenzte Rechenkapazität

nur einige der Hyperparameter getestet werden konnten.

Nach dem erfolgreichen Trainieren aller Modelle mit den verschiedenen Hyperparameter können die Ergebnisse ausgewertet werden. Dafür bedarf es noch einer Definition zur Messung dieser Ergebnisse.

Für jeden gegebenen Klassifikator und jede Instanz gibt es vier mögliche Ergebnisse. Wenn ein positiver Fall korrekt als positiv identifiziert wird, spricht man von einem wahren Positiv. Wird dieser positive Fall fälschlicherweise als negativ eingestuft, wird er als falsch Negativ bezeichnet. Umgekehrt wird eine negative Instanz, die korrekt als negativ eingestuft wird, als wahres Negativ bezeichnet. Zur Messung der Resultate in Kapitel 6 wurden die folgenden Maße verwendet. Dabei sind hier die wahren Positiv als $T_p$, die falsch Positiv als $F_p$ und alle positiven als $P$ gekennzeichnet. Damit gilt:

$$\begin{aligned} precision &= \frac{T_p}{T_p + F_p} \\ recall &= \frac{T_p}{P} \\ F1 &= \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \end{aligned}$$

Aus der Menge der wahren und falsch Positiven und den Performance Maßen können auch Grafiken erstellt werden, um die Modelle zu analysieren. Eine davon ist die *receiver operating characteristic* (ROC). Auf dieser Grafik wird die wahre Positiv Rate auf der Ordinate und die falsch Positiv Rate auf der Abszisse dargestellt. Dabei wird ein Modell mit verschiedenen Klassifizierungsgrenzen ausgewertet und pro Grenzwert ein Punkt als ROC Kurve erstellt. Ein Punkt bei $(0, 0)$ zeigt einen Klassifizierer ohne falsch Positiv Fehler aber auch ohne wahre Positive. Ein Punkt auf $(1, 1)$ ist damit das Gegenteil vom Punkt im Ursprung. Und ein Punkt in $(0, 1)$ würde eine perfekte Klassifizierung bedeuten. Zusammengefasst bedeutet dies, dass alle Punkte links von der Diagonale von $(0, 0)$ nach $(1, 1)$ einen Klassifizierer zeigen, welcher besser ist als zufälliges Raten. Ein großer Vorteil von ROC-Diagrammen ist, dass sie ohne Berücksichtigung von Klassenverteilungen oder Fehlerkosten die Visualisierung eines Klassifikators ermöglichen [63]. Damit sind die ROC Grafiken unsensibel auf Veränderungen in der Klassenverteilung. Dieser Aspekt ist für die Analyse der Performance zwischen den Zielen KFZ, Sach und Gesundheit sehr wichtig, da damit ein Vergleich untereinander möglich wird. Aus der ROC-Kurve kann gleichzeitig noch eine weitere Statistik berechnet werden, dabei handelt es sich um die *area under the curve* (AUC). Der AUC besitzt eine bedeutende statistische Eigenschaft: Er entspricht der Wahrscheinlichkeit, dass ein Klassifikator eine zufällig ausgewählte positive Instanz höher bewertet als eine zufällig ausgewählte negative Instanz. Mit anderen Worten, der AUC gibt an, wie gut ein Klassifikator in der Lage ist, positive von negativen Fällen zu unterscheiden. Je höher der AUC-Wert, desto größer die Wahrscheinlichkeit, dass der Klassifikator eine positive Instanz im Vergleich zu einer negativen Instanz korrekt priorisiert.

Um in weiterer Folge den Einfluss verschiedener Klassenungleichgewichte zu zeigen, benötigt es einer anderen Grafik. Dabei kann die Veränderung zwischen Precision und Recall für verschiedene Klassifikationsgrenzen betrachtet werden.

# 6. Resultate

In diesem Kapitel werden alle relevanten Ergebnisse für diese Arbeit präsentiert und deren Ausführung erläutert. Als erste Resultate werden an dieser Stelle zwei ROC Graphen gezeigt. In Abbildung 6.1 sind die ROC Kurven des Catboost Algorithmus für das Ziel KFZ zu sehen. Dieser Plot für diesen Algorithmus wurde stellvertretend für alle Modelle mit dem Ziel als KFZ ausgewählt. Tatsächlich unterscheiden sich die ROC Grafiken nur sehr gering bei der Anwendung von verschiedenen Algorithmen. Hinzu kommt, dass der ROC sich auch bei dem Ziel Sach und Gesundheit relativ wenig unterscheidet. Der Grund dafür ist die im obigen Kaptiel beschriebene Eigenschaften, dass ein ROC Graph nicht in bei Klassenungleichgewichten unterscheidet. Eine somit sehr viel geringeres Kaufverhalten bei Gesundheit wird somit nicht erfasst.

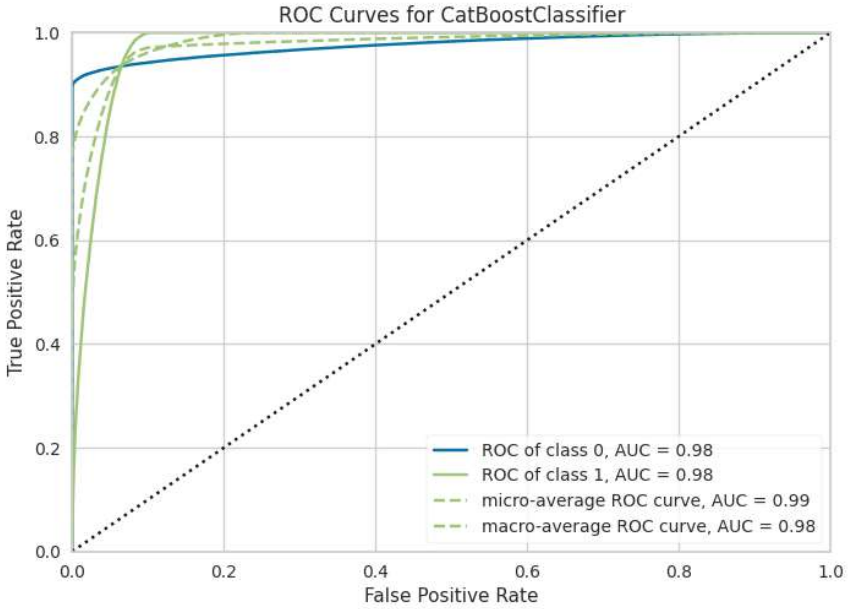


Abbildung 6.1.: ROC von Catboost Algorithmus auf dem Ziel KFZ mit einem Feature Select von 0,25

In der Abbildung 6.2 wird die ROC für das Ziel Gesundheit mit der Dimensionsreduktionsmethode PCA gezeigt. Auch hier wird wieder, aus dem selben Grund wie zuvor, stellvertretend der Catboost Algorithmus gezeigt. Damit ist ersichtlich, dass PCA sich nicht vorteilhaft für diese Klassifizierung auswirkt. Die einzige Ausnahme dafür ist die QDA Methode. Darauf wird später noch genauer eingegangen.

Mit Hilfe der ROC Abbildungen konnte gezeigt werden, dass alle gewählten Modelle aus

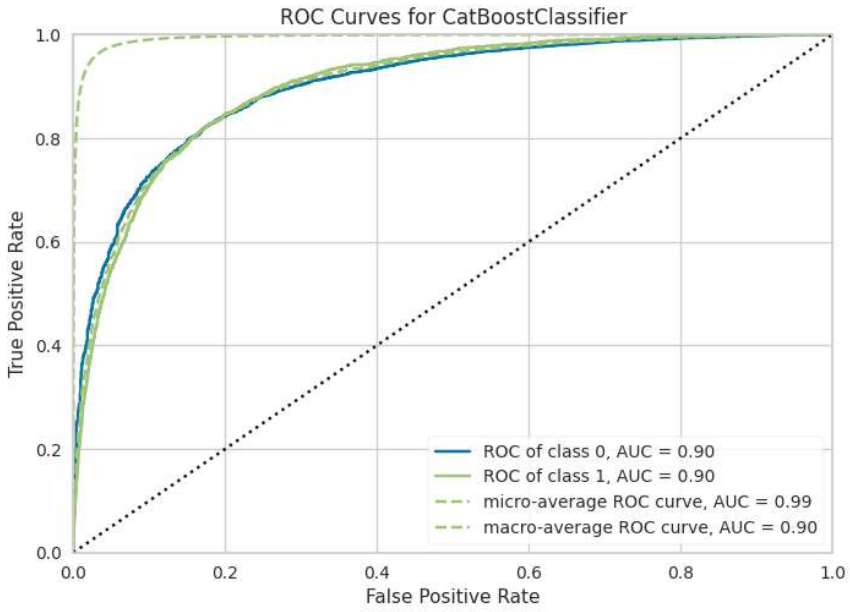


Abbildung 6.2.: ROC von Catboost Algorithmus auf dem Ziel Gesundheit mit einem PCA Feature Select

Kapitel 3 in der Lage sind, eine Klassifizierung für die 3 gewählten Risikoprofile zu erstellen, welche um einiges besser als Zufallsraten (Diagonale des ROC) sind.

Als nächste Grafik wurden die Precision-Recall Plots untersucht, zu sehen in Abbildung 6.3 und 6.4. Auch hier wurde wieder stellvertretend die Catboost Methode gewählt, da auch hier wieder die Modelle sich nur gering voneinander unterscheiden. Der Hauptgrund dieses Plots ist es, zu untersuchen wie sich das starke Klassenungleichgewicht auf die Modelle auswirkt. Zunächst ist die Kurve in Abbildung 6.3, wo das Ziel das KFZ Risikoprofil war, eine mit einem sehr hohen Precision Durchschnitt. Da auch bei einem sehr hohen Recall, die Precision mit über 0.6 einen guten Wert aufweist, wird sich auch der F1-Score bei diesem Modell in einem attraktiven Niveau bewegen.

Ein Blick auf die Precision-Recall von einem beliebigen Modell, welches das Gesundheits Risikoprofil als Ziel hat, zeigt dass sich hier nur schwierig eine hohe Precision durchsetzen lassen kann. Da der Recall vergleichsweise stabil ist, bedeutet dies, dass hier viele falsch Positiv klassifiziert wurden.

Die beiden Grafiken für ROC und Precision-Recall zeigen in Summe, dass das Klassenungleichgewicht die Modell-Performance stark beeinflusst. Um näher auf die einzelnen Modelle einzugehen, um schließlich das Bestmögliche auswählen zu können, werden im Folgenden die Performance Maße als Werte in Tabellen gezeigt. Die Aufgabe eines erfolgreichen *Recommender Systems* ist es so gut wie möglich wahre Positiv zu klassifizieren ohne zugleich die Menge der tatsächlich Positiven zu verfehlen. Um dies zu erzielen, wird in dieser Arbeit jenes Modell als bestes ausgewählt, welches den höchsten F1-Score erzielt. Dabei sei erlaubt, dass ein Modell nur für ein einziges Risikoprofil optimal ist.

Da durch das Klassenungleichgewicht in den drei Risikoprofilen sehr unterschiedlich ist,

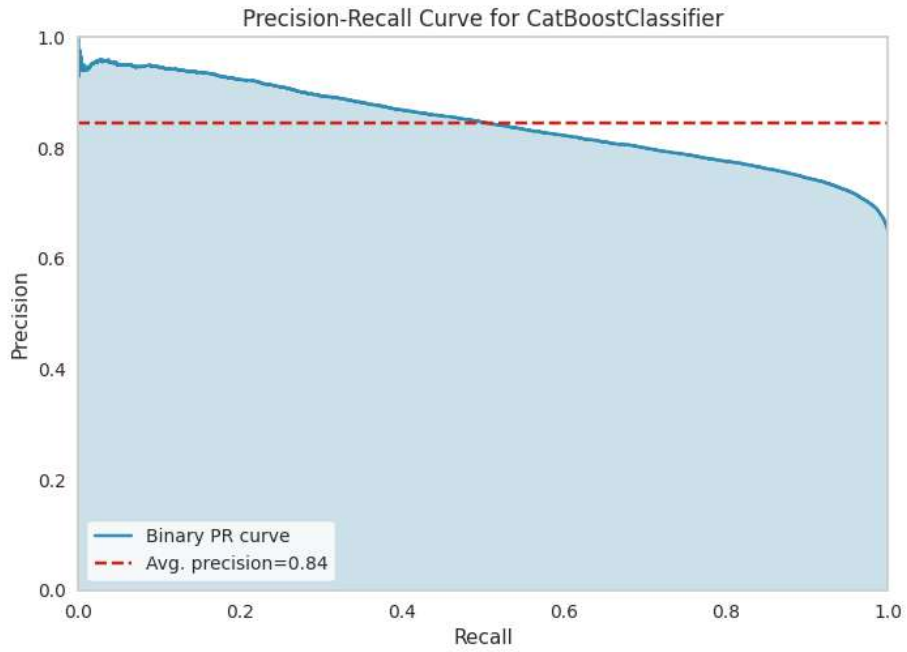


Abbildung 6.3.: Precision-Recall Kurve des Catboost Algorithmus auf dem Ziel KFZ ohne Feature Select

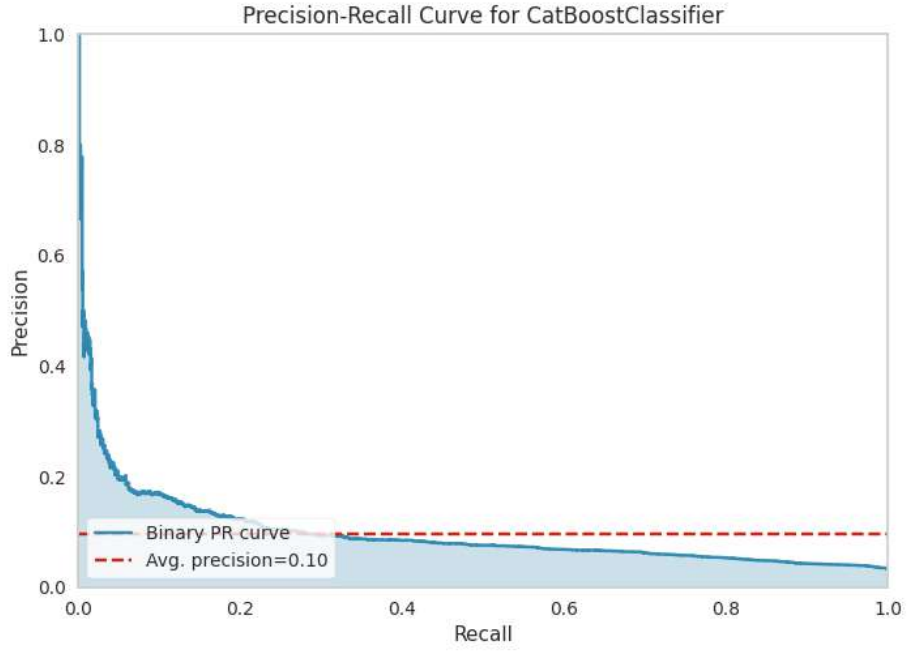


Abbildung 6.4.: Precision-Recall Kurve des Catboost Algorithmus auf dem Ziel Gesundheit mit einem Feature Select von 0,25

werden die Resultate der Modelle für jede der ausgewählten Dimensionsreduktionsmethoden in den folgenden Unterkapitel 6.1 bis 6.3 aufgeteilt. Dabei wird jeweils zuerst das Resultat bzgl. dem F1-Score auf der gesamten Testmenge gezeigt. Anschließend werden die Resultate auf die Menge der positiven bzw. negativen („gekauft“= 1 und „gekauft“= 0) getrennt gezeigt. Dadurch sollen die Ergebnisse transparent und verständlich aufgelistet werden. Zudem wird dadurch gezeigt, wo die Stärke und wo die Schwäche der einzelnen Modelle liegt. Um die Lesbarkeit zu erhöhen wird in den Tabellen mit den Resultaten immer jenes Modell mit dem höchsten Score hervorgehoben und jene mit dem niedrigsten bzw. einem Score unter 0.50 leicht ausgegraut.

## 6.1. KFZ Privat

Der erste Blick fällt auf den F1-Score für die gesamte Testmenge für die verschiedenen Modelle auf den Szenarien: PCA Reduktion, Feature Select (FS) von 0.75, 0.5 und 0.25 sowie einem mit allen Merkmalen nach der Vorverarbeitungsphase (siehe Tabelle 6.1). Der F1-Score ist bei fast allen Szenarien über einem Wert von 0.70 und bei den Spitzenreitern sogar um die 0.90 . Auffallend ist der schlechte Wert bei NB mit allen Merkmalen und die QDA, welche nur bei PCA und einem FS von 0.25 gute Ergebnisse erzielt. Es zeigt sich auch, dass die PCA Methode bei allen außer QDA das Ergebnis verschlechtert. Das beste Modell bei PCA sind die NN, welche bei allen Szenarien ähnliche Ergebnisse erzielen und somit von einer gewissen Stabilität zeugen. Weiters ist ersichtlich, dass die Baum-Methoden (DT, RF, XGB, LGBM, CatB) immer an erster bzw. bei PCA an zweiter Stelle stehen.

| | NB | LR | LDA | QDA | DT | RF | XGB | LGBM | CatB | NN |
|---|---|---|---|---|---|---|---|---|---|---|
| Alle Merkmale | 0.22 | 0.86 | 0.86 | 0.31 | 0.85 | 0.89 | 0.90 | 0.90 | **0.91** | 0.87 |
| PCA | 0.72 | 0.75 | 0.75 | 0.73 | 0.70 | 0.78 | 0.79 | 0.77 | 0.82 | **0.86** |
| #Merkmale 0,75 | 0.80 | 0.87 | 0.85 | 0.32 | 0.87 | **0.91** | 0.87 | 0.87 | 0.88 | 0.89 |
| #Merkmale 0,5 | 0.83 | 0.87 | 0.84 | 0.25 | 0.84 | **0.89** | **0.89** | **0.89** | **0.89** | 0.88 |
| #Merkmale 0,25 | 0.86 | 0.86 | 0.84 | 0.86 | 0.84 | 0.88 | **0.89** | **0.89** | **0.89** | 0.88 |

Tabelle 6.1.: Ziel KFZ Privat, F1 Score auf der gesamten Testmenge

Das Ziel dieser Arbeit ist es aus den ausgewählten Modellen und Szenarien das Bestmöglichste auszuwählen, um Cross-Selling Möglichkeiten gezielt einzusetzen. Dabei gilt ein besonderes Interesse an der Leistung der Modelle für die tatsächlich gekauften Produkte im jeweiligen Risikoprofil. Aus diesem Grund zeigt die Tabelle 6.2 die Ergebnisse für den F1-Score auf der Testmenge mit den tatsächlich gekauften Produkten. Hier sind alle Resultate um ca. 0.10 schlechter als auf der gesamten Testmenge. Dieser Effekt war zu erwarten, da es wie im Kapitel 5 darauf hingewiesen, ein Klassenungleichgewicht gibt. Den besten Score erzielt nun der RF mit 0.85 nur sehr knapp über dem CatB welcher auf der gesamten Testmenge der Sieger war. Im Grunde sind die besten Modelle je Szenario dieselben Algorithmen. Die QDA erzielt auf dieser Testmenge bessere Resultate als auf der gesamten Menge. Damit könnte es sein, dass QDA eine ähnliche Precision-Recall auf der Menge „gekauft“= 1 erzielt als wie auf „gekauft“= 0. Gleichzeitig könnte das auch bedeuten, dass die restlichen

Algorithmen ein stärkeres Ungleichgewicht zwischen Precision und Recall auf der Menge der gekauften Produkte aufweisen.

| | NB | LR | LDA | QDA | DT | RF | XGB | LGBM | CatB | NN |
|---|---|---|---|---|---|---|---|---|---|---|
| Alle Merkmale | 0.29 | 0.77 | 0.77 | 0.37 | 0.75 | 0.82 | 0.83 | 0.83 | **0.84** | 0.80 |
| PCA | 0.55 | 0.58 | 0.66 | 0.65 | 0.51 | 0.63 | 0.65 | 0.62 | 0.69 | **0.77** |
| #Merkmale 0,75 | 0.69 | 0.79 | 0.77 | 0.55 | 0.78 | **0.85** | 0.79 | 0.79 | 0.80 | 0.82 |
| #Merkmale 0,5 | 0.73 | 0.78 | 0.77 | 0.51 | 0.73 | **0.82** | **0.82** | **0.82** | **0.82** | 0.81 |
| #Merkmale 0,25 | 0.77 | 0.78 | 0.76 | 0.77 | 0.73 | 0.80 | **0.82** | **0.82** | 0.81 | 0.81 |

Tabelle 6.2.: Ziel KFZ Privat, F1 Score auf der Testmenge mit „gekauft“= 1

Um den Verdacht der aus Tabelle 6.2 gewonnenen Einsicht zu bestätigen zeigen die Tabellen 6.3 und 6.4 den Precision und Recall Wert für jedes Modell und jedes Szenario.

| | NB | | LR | | LDA | | QDA | | DT | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.17 | **1.00** | 0.64 | 0.99 | 0.63 | **1.00** | 0.19 | **1.00** | 0.75 | 0.75 |
| PCA | 0.47 | 0.65 | 0.53 | 0.65 | 0.52 | 0.66 | 0.46 | **0.74** | 0.43 | 0.61 |
| #Merkmale 0,75 | 0.53 | 0.97 | 0.67 | 0.95 | 0.62 | **1.00** | 0.19 | **1.00** | 0.68 | 0.92 |
| #Merkmale 0,5 | 0.58 | 0.97 | 0.66 | 0.97 | 0.60 | **1.00** | 0.17 | **1.00** | 0.73 | 0.73 |
| #Merkmale 0,25 | 0.65 | 0.94 | 0.64 | 0.99 | 0.59 | **1.00** | 0.65 | 0.93 | 0.73 | 0.72 |
| | RF | | XGB | | LGBM | | CatB | | NN | |
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.76 | 0.90 | 0.78 | 0.89 | 0.75 | 0.93 | 0.79 | 0.90 | 0.66 | **0.99** |
| PCA | 0.59 | 0.68 | 0.60 | 0.71 | 0.56 | 0.70 | 0.67 | 0.73 | 0.71 | **0.85** |
| #Merkmale 0,75 | 0.75 | 0.99 | 0.65 | **1.00** | 0.65 | **1.00** | 0.67 | 0.99 | 0.71 | 0.97 |
| #Merkmale 0,5 | 0.74 | 0.91 | 0.75 | 0.90 | 0.74 | 0.92 | 0.75 | 0.89 | 0.68 | **0.99** |
| #Merkmale 0,25 | 0.74 | 0.87 | 0.74 | 0.91 | 0.74 | 0.92 | 0.74 | 0.90 | 0.70 | **0.98** |

Tabelle 6.3.: Ziel KFZ Privat, Precision und Recall auf der Testmenge mit „gekauft“= 1

Ein erster Blick auf die höchsten Werte pro Szenario zeigen, dass immer mindestens ein Modell einen perfekten Recall oder Precision erzielt. Zugleich sind sehr hohe Recall Werte bei QDA einem sehr niedrigen Precision Wert gegenüber gestellt. Dieser Effekt ist auf der Menge „gekauft“= 1 verdreht wir auf „gekauft“= 0. Damit lässt sich sagen, dass das Modell bei seinen Vorhersagen für Klasse 1 konservativ ist, um falsch Positiv Ergebnisse zu vermeiden, und bei den Vorhersagen für Klasse 0 großzügig ist, was zu mehr falsch Positiv Ergebnissen führt, aber die meisten tatsächlichen Instanzen erfasst. Auf der Testmenge „gekauft“= 0 sind zudem auch beide Scores um einiges höher, dies ist die direkte Folge des Klassenungleichgewichts welches sich auf den Träger der Testmenge niederschlägt.

Auf dem Szenario mit PCA zeigt sich ein sehr ausgeglichenes Verhalten zwischen Precision und Recall, und zwar auf beiden Klassen. Wenn bestimmte Merkmale im Datensatz für eine Klasse aussagekräftiger sind als für die andere, kann ein Modell diese Merkmale für diese Klasse bevorzugen. Wenn beispielsweise Merkmale, die bei der Identifizierung

| | NB | | LR | | LDA | | QDA | | DT | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.99 | 0.08 | 0.99 | 0.89 | **1.00** | 0.89 | **1.00** | 0.18 | 0.95 | 0.95 |
| PCA | **0.99** | 0.86 | 0.93 | 0.89 | 0.93 | 0.89 | 0.95 | 0.84 | 0.92 | 0.85 |
| #Merkmale 0,75 | 0.99 | 0.84 | 0.99 | 0.91 | **1.00** | 0.88 | **1.00** | 0.19 | 0.98 | 0.92 |
| #Merkmale 0,5 | 0.99 | 0.87 | 0.99 | 0.90 | **1.00** | 0.88 | 0.99 | 0.11 | 0.95 | 0.95 |
| #Merkmale 0,25 | 0.99 | 0.90 | 0.99 | 0.90 | **1.00** | 0.87 | 0.99 | 0.90 | 0.95 | 0.95 |
| | RF | | XGB | | LGBM | | CatB | | NN | |
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.98 | 0.95 | 0.98 | 0.95 | 0.99 | 0.94 | 0.98 | 0.95 | **1.00** | 0.90 |
| PCA | 0.94 | 0.91 | 0.94 | 0.91 | 0.94 | 0.90 | 0.95 | 0.93 | **0.97** | 0.93 |
| #Merkmale 0,75 | 0.99 | 0.94 | **1.00** | 0.90 | **1.00** | 0.90 | **1.00** | 0.91 | 0.99 | 0.92 |
| #Merkmale 0,5 | 0.98 | 0.94 | 0.98 | 0.94 | 0.98 | 0.94 | 0.98 | 0.94 | **1.00** | 0.91 |
| #Merkmale 0,25 | 0.97 | 0.94 | 0.98 | 0.94 | 0.98 | 0.94 | 0.98 | 0.94 | **0.99** | 0.92 |

Tabelle 6.4.: Ziel KFZ Privat, Precision und Recall auf der Testmenge mit „gekauft“= 0

von Klasse 0 helfen, stärker oder eindeutiger sind, wird das Modell einen höheren Recall für Klasse 0 haben, aber die Precision kann aufgrund von Überschneidungen mit Klasse 1 leiden.

## 6.2. Sach Privat

Das Ergebnis für den F1-Score auf der gesamten Testmenge sei in Tabelle 6.5 dargestellt. Das Resultat zeigt eine große Ähnlichkeit mit dem KFZ Privat Modellergebnis, auch wenn die absoluten Werte etwas geringer ausfallen. QDA und NB schneiden jeweils bei einzelnen Szenarien wesentlich schlechter ab als die restlichen Modelle und PCA zeigt wieder einen Schwachpunkt in den Szenarien auf. Auch hier ist der Algorithmus mit dem höchsten Score ein Baum-Modell und zwar der LGBM bei dem Szenario mit allen Merkmalen, gefolgt von CatB mit einem FS von 0.75 . Die NN zeigen wieder eine Stabilität über alle Szenarien hinweg und sind bei zwei Szenarien an erster Stelle.

| | NB | LR | LDA | QDA | DT | RF | XGB | LGBM | CatB | NN |
|---|---|---|---|---|---|---|---|---|---|---|
| Alle Merkmale | 0.16 | 0.71 | 0.69 | 0.46 | 0.78 | 0.77 | 0.67 | **0.86** | 0.69 | 0.77 |
| PCA | 0.63 | 0.70 | 0.69 | 0.65 | 0.63 | 0.71 | 0.73 | 0.69 | 0.73 | **0.76** |
| #Merkmale 0,75 | 0.71 | 0.66 | 0.63 | 0.39 | 0.73 | 0.75 | 0.68 | 0.69 | **0.81** | 0.79 |
| #Merkmale 0,5 | 0.67 | 0.72 | 0.70 | 0.70 | 0.74 | 0.76 | 0.66 | **0.80** | 0.67 | 0.78 |
| #Merkmale 0,25 | 0.68 | 0.70 | 0.68 | 0.71 | 0.66 | 0.70 | 0.69 | 0.67 | 0.68 | **0.77** |

Tabelle 6.5.: Ziel Sach Privat, F1 Score auf der gesamten Testmenge

Aus dem selben Grund wie bei KFZ Privat erklärt, gilt dem Ergebniswert für den F1-Score auf der Klasse 1 ein besonderes Interesse. In Tabelle 6.6 sind die Resultate dazu präsentiert. Sofort ist ersichtlich, dass die Mehrheit der Szenarien einen F1-Score unter 0.50

haben. Der Hauptgrund hierfür könnte das extremere Klassenungleichgewicht gegenüber dem KFZ Privat Profil sein, daraus ist direkt der Träger auf der Testmenge mit „gekauft“= 1 um einiges kleiner. Trotzdem setzt sich der LGBM Algorithmus mit einem F1 Score von 0.73 als Bestes, wie auf der gesamten Testmenge, durch. Auch hier zeigt das NN den stabilsten Score über alle Szenarien hinweg auf und zudem erzielt dieses immer einen Wert über 0.50 und ist die beste Wahl bei den Szenarien PCA und FS 0.25 . Im Vergleich zu den Resultaten aus KFZ Privat fällt auf, dass die QDA sich auf der Gesamtmenge zu Testmenge mit Klasse 1 verschlechtert. Damit ist es möglich, dass auch beim QDA kein stabiles Precision-Recall Verhältnis erzielt wurde. Um diesen Verdacht weiter zu bestätigen, werden im Folgenden wieder die Resultate für die Precision sowie Recall für die Modelle und Szenarien in den Tabellen 6.7 und 6.8 aufgelistet.

| | NB | LR | LDA | QDA | DT | RF | XGB | LGBM | CatB | NN |
|---|---|---|---|---|---|---|---|---|---|---|
| Alle Merkmale | 0.15 | 0.50 | 0.48 | 0.20 | 0.58 | 0.57 | 0.45 | **0.73** | 0.48 | 0.59 |
| PCA | 0.34 | 0.46 | 0.45 | 0.39 | 0.34 | 0.47 | 0.50 | 0.44 | 0.51 | **0.56** |
| #Merkmale 0,75 | 0.48 | 0.39 | 0.37 | 0.18 | 0.51 | 0.55 | 0.46 | 0.47 | **0.64** | 0.62 |
| #Merkmale 0,5 | 0.44 | 0.51 | 0.48 | 0.48 | 0.53 | 0.57 | 0.44 | **0.63** | 0.45 | 0.60 |
| #Merkmale 0,25 | 0.46 | 0.49 | 0.46 | 0.50 | 0.44 | 0.49 | 0.47 | 0.45 | 0.46 | **0.59** |

Tabelle 6.6.: Ziel Sach Privat, F1 Score auf der Testmenge mit „gekauft“= 1

| | NB | | LR | | LDA | | QDA | | DT | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.08 | **0.98** | 0.33 | 0.96 | 0.32 | **0.98** | 0.12 | 0.70 | 0.63 | 0.55 |
| PCA | 0.25 | 0.52 | 0.36 | 0.65 | 0.34 | 0.64 | 0.27 | **0.67** | 0.25 | 0.56 |
| #Merkmale 0,75 | 0.36 | 0.71 | 0.29 | 0.58 | 0.23 | 0.64 | 0.10 | **0.78** | 0.39 | 0.75 |
| #Merkmale 0,5 | 0.29 | **0.95** | 0.36 | 0.86 | 0.32 | **0.95** | 0.32 | 0.92 | 0.45 | 0.64 |
| #Merkmale 0,25 | 0.31 | 0.92 | 0.34 | 0.92 | 0.30 | **0.99** | 0.34 | 0.93 | 0.27 | **0.99** |
| | RF | | XGB | | LGBM | | CatB | | NN | |
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.80 | 0.45 | 0.29 | **0.99** | 0.85 | 0.64 | 0.32 | 0.96 | 0.54 | 0.63 |
| PCA | 0.42 | 0.54 | 0.44 | 0.57 | 0.38 | 0.54 | 0.46 | 0.58 | **0.63** | 0.50 |
| #Merkmale 0,75 | 0.41 | 0.85 | 0.30 | **0.99** | 0.31 | **0.99** | 0.77 | 0.54 | 0.59 | 0.64 |
| #Merkmale 0,5 | 0.46 | 0.73 | 0.28 | **0.99** | 0.74 | 0.54 | 0.29 | 0.98 | 0.55 | 0.67 |
| #Merkmale 0,25 | 0.33 | 0.92 | 0.31 | 0.93 | 0.29 | **0.99** | 0.30 | **0.99** | 0.51 | 0.69 |

Tabelle 6.7.: Ziel Sach Privat, Precision und Recall auf der Testmenge mit „gekauft“= 1

Es ist sofort ersichtlich, dass in Tabelle 6.7 ein starkes Ungleichgewicht zwischen Precision und Recall herrscht. Nur die NN zeichnen ein stabiles Bild mit einem ausgeglichenen Precision und Recall von 0.50 bis 0.70 in allen Szenarien. Nahezu alle anderen Szenarien haben eine Precision unter 0.50 und sogar unter 0.20 . Lediglich die DT, RF, LGBM und CatB zeigen im Szenario mit allen Merkmalen einen höheren Precision als Recall. Im Allgemeinen schneiden die Baum-Modelle damit wieder relativ gut ab (im Vergleich zu den

anderen Algorithmen).

| | NB | | LR | | LDA | | QDA | | DT | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.98 | 0.09 | **0.99** | 0.84 | **0.99** | 0.83 | 0.96 | 0.56 | 0.96 | 0.97 |
| PCA | 0.96 | 0.87 | **0.97** | 0.90 | **0.97** | 0.90 | **0.97** | 0.85 | 0.96 | 0.86 |
| #Merkmale 0,75 | 0.97 | 0.89 | 0.96 | 0.88 | 0.96 | 0.82 | 0.96 | 0.43 | **0.98** | 0.90 |
| #Merkmale 0,5 | **0.99** | 0.80 | 0.98 | 0.87 | **0.99** | 0.83 | **0.99** | 0.84 | 0.97 | 0.93 |
| #Merkmale 0,25 | 0.99 | 0.83 | 0.99 | 0.85 | **1.00** | 0.81 | 0.99 | 0.85 | 0.99 | 0.79 |
| | RF | | XGB | | LGBM | | CatB | | NN | |
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.95 | **0.99** | **0.99** | 0.80 | 0.97 | **0.99** | **0.99** | 0.83 | 0.97 | 0.96 |
| PCA | 0.96 | 0.94 | 0.96 | 0.94 | 0.96 | 0.92 | 0.96 | 0.94 | 0.96 | **0.98** |
| #Merkmale 0,75 | 0.99 | 0.90 | **1.00** | 0.81 | 0.99 | 0.82 | 0.96 | 0.99 | 0.97 | 0.96 |
| #Merkmale 0,5 | 0.98 | 0.93 | **1.00** | 0.79 | 0.96 | 0.98 | 0.99 | 0.80 | 0.97 | 0.95 |
| #Merkmale 0,25 | 0.99 | 0.84 | 0.99 | 0.83 | **1.00** | 0.79 | 0.99 | 0.81 | 0.97 | 0.95 |

Tabelle 6.8.: Ziel Sach Privat, Precision und Recall auf der Testmenge mit „gekauft“= 0

Zu den Werten in Tabelle 6.8 kann keine große neue Erkenntnis gezogen werden außer, dass die Algorithmen bei einer großen Testmenge im Allgemeinen sehr gut abschneiden und ein hoher Träger bei der Klasse 0 dies begünstigt. Es sei auch auf den extrem niedrigen Wert bei NB mit allen Merkmalen hingewiesen, dieser Fall zeigt sich quer durch alle Szenarien und Risikoprofile. Damit gilt, dass NB diese Datenmenge mit sehr vielen verschiedenen Merkmalen nicht gut klassifizieren kann.

## 6.3. Gesundheit

Zu guter Letzt seien die Resultate (in gleicher Reihenfolge wie bisher) für das Risikoprofil Gesundheit in den folgenden Tabellen angeführt. Da das größte Klassenungleichgewicht bei diesem Risikoprofil zu finden ist, und alle anderen nicht gezeigten Risikoprofile der Allianz ein noch größeres Ungleichgewicht beinhalten, wird mit diesem Resultat ein Grenzfall gezeigt an dem nahezu alle Modelle scheitern die gegebene Testmengen zu klassifizieren.

| | NB | LR | LDA | QDA | DT | RF | XGB | LGBM | CatB | NN |
|---|---|---|---|---|---|---|---|---|---|---|
| Alle Merkmale | 0.12 | 0.54 | 0.53 | 0.40 | 0.61 | **0.62** | 0.55 | 0.60 | 0.61 | 0.61 |
| PCA | 0.49 | 0.52 | 0.52 | 0.50 | 0.52 | 0.58 | 0.59 | 0.52 | 0.58 | **0.60** |
| #Merkmale 0,75 | 0.47 | 0.57 | 0.56 | 0.51 | 0.54 | **0.62** | 0.49 | 0.59 | 0.55 | **0.62** |
| #Merkmale 0,5 | 0.50 | 0.56 | 0.54 | 0.55 | 0.54 | 0.56 | **0.61** | 0.59 | 0.58 | 0.60 |
| #Merkmale 0,25 | 0.48 | 0.55 | 0.53 | 0.55 | 0.52 | 0.54 | 0.47 | 0.50 | 0.48 | **0.59** |

Tabelle 6.9.: Ziel Gesundheit, F1 Score auf der gesamten Testmenge

Das Ergebnis auf der gesamten Testmenge zeigt, im Vergleich zu den obigen Werten, sehr viele F1-Scores um 0.50 und darunter. Den Platz um den besten F1-Score teilen sich die RF

und NN. Interessant zu sehen ist, dass sich nun auf dieser Menge mit einer geringen Präsenz der positiven Klasse der Unterschied der Szenarien nicht mehr so stark miteinbringt. Nur der NB Klassifizierer zeigt, wie zuvor vermerkt, bei dem Szenario mit allen Merkmalen ein sehr niedrigen F1-Score.

| | NB | LR | LDA | QDA | DT | RF | XGB | LGBM | CatB | NN |
|---|---|---|---|---|---|---|---|---|---|---|
| Alle Merkmale | 0.02 | 0.13 | 0.12 | 0.03 | 0.23 | 0.25 | 0.15 | 0.21 | 0.24 | 0.24 |
| PCA | 0.05 | 0.09 | 0.09 | 0.07 | 0.08 | 0.16 | 0.18 | 0.09 | 0.18 | 0.22 |
| #Merkmale 0,75 | 0.06 | 0.17 | 0.15 | 0.09 | 0.12 | 0.25 | 0.08 | 0.20 | 0.14 | 0.25 |
| #Merkmale 0,5 | 0.07 | 0.15 | 0.13 | 0.14 | 0.13 | 0.16 | 0.22 | 0.20 | 0.16 | 0.21 |
| #Merkmale 0,25 | 0.01 | 0.14 | 0.12 | 0.14 | 0.10 | 0.13 | 0.07 | 0.09 | 0.07 | 0.20 |

Tabelle 6.10.: Ziel Gesundheit, F1 Score auf der Testmenge mit „gekauft“= 1

Tabelle 6.10 zeigt sehr gut, wie sich ein verschwindend geringer Träger der Klasse 1 auf den F1-Score auswirkt. Nicht ein einziges Szenario erzielt einen F1-Score über 0.50, der höchste Score wird von den NN mit 0.25 erzielt. Die Werte auf der gesamten Testmenge (Tabelle 6.9) sind somit nur durch das hervorragende Training auf der negativen Klasse verfälscht.

Um diese Behauptung weiter zu untersuchen, sei wieder auf die Resultate der Precision und Recall Werte auf den Testmengen mit „gekauft“= 1 (Tabelle 6.11) und „gekauft“= 0 (Tabelle 6.12) verwiesen.

Wie auch schon bei dem Ziel Sach Privat ist hier der Recall weiterhin um einiges höher als wie die Precision. Dabei zeigt sich, dass durch das größere Klassenungleichgewicht auch das Verhältnis von Precision zu Recall verschlechtert. Dieser Effekt war bereits in dem Precision-Recall Graphen (Abb. 6.4) zu sehen.

| | NB | | LR | | LDA | | QDA | | DT | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.01 | **0.98** | 0.07 | 0.90 | 0.07 | 0.88 | 0.02 | 0.74 | 0.16 | 0.42 |
| PCA | 0.03 | 0.43 | 0.05 | 0.53 | 0.05 | 0.52 | 0.04 | **0.62** | 0.04 | 0.35 |
| #Merkmale 0,75 | 0.03 | 0.72 | 0.09 | 0.74 | 0.08 | **0.83** | 0.05 | 0.75 | 0.06 | 0.60 |
| #Merkmale 0,5 | 0.04 | 0.67 | 0.08 | 0.78 | 0.07 | **0.85** | 0.07 | 0.76 | 0.07 | 0.73 |
| #Merkmale 0,25 | 0.05 | 0.73 | 0.08 | 0.82 | 0.07 | **0.88** | 0.08 | 0.82 | 0.06 | 0.73 |
| | RF | | XGB | | LGBM | | CatB | | NN | |
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.28 | 0.22 | 0.08 | **0.96** | 0.78 | 0.12 | 0.14 | 0.84 | 0.17 | 0.39 |
| PCA | 0.12 | 0.23 | 0.14 | 0.27 | 0.05 | **0.50** | 0.11 | 0.44 | 0.14 | 0.49 |
| #Merkmale 0,75 | 0.21 | 0.30 | 0.04 | **0.98** | 0.12 | 0.66 | 0.07 | 0.90 | 0.17 | 0.50 |
| #Merkmale 0,5 | 0.09 | **0.76** | 0.42 | 0.15 | 0.12 | 0.57 | 0.54 | 0.09 | 0.13 | 0.60 |
| #Merkmale 0,25 | 0.07 | 0.81 | 0.04 | **0.97** | 0.04 | 0.91 | 0.04 | **0.97** | 0.12 | 0.71 |

Tabelle 6.11.: Ziel Gesundheit, Precision und Recall auf der Testmenge mit „gekauft“= 1

In der Tabelle mit den Resultaten zu Precision und Recall auf der Testmenge mit „ge-

kauft“= 0 zeichnet sich auch wieder ein ähnliches Bild wie bereits in den Ergebnissen zu Sach Privat zu sehen war. Durch den größeren Träger der Klasse 0 sind die beiden Maße noch um ein paar Punkte gestiegen. Alle Szenarien erzielen auf dieser Testmenge hervorragende Resultate, bis auf den NB Fall mit allen Merkmalen. Wie auch zuvor ist hier, wenn auch in einem geringeren Ausmaß, zu sehen dass der Recall Wert tendenziell unter dem Precision liegt.

| | NB | | LR | | LDA | | QDA | | DT | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | **0.99** | 0.12 | **0.99** | 0.89 | **0.99** | 0.89 | **0.99** | 0.63 | **0.99** | 0.98 |
| PCA | **0.99** | 0.86 | **0.99** | 0.91 | **0.99** | 0.91 | **0.99** | 0.86 | **0.99** | 0.93 |
| #Merkmale 0,75 | **0.99** | 0.79 | **0.99** | 0.94 | **0.99** | 0.92 | **0.99** | 0.87 | **0.99** | 0.92 |
| #Merkmale 0,5 | **0.99** | 0.85 | **0.99** | 0.92 | **0.99** | 0.90 | **0.99** | 0.92 | **0.99** | 0.92 |
| #Merkmale 0,25 | **0.99** | 0.88 | **0.99** | 0.91 | **0.99** | 0.89 | **0.99** | 0.91 | **0.99** | 0.89 |
| | RF | | XGB | | LGBM | | CatB | | NN | |
| | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec | Prec | Rec |
| Alle Merkmale | 0.99 | 0.99 | **1.00** | 0.90 | 0.99 | **1.00** | 0.99 | 0.95 | 0.99 | 0.98 |
| PCA | 0.99 | 0.98 | 0.99 | 0.98 | 0.99 | 0.91 | 0.99 | 0.97 | **1.00** | 0.97 |
| #Merkmale 0,75 | 0.99 | 0.99 | **1.00** | 0.81 | 0.99 | 0.96 | 0.99 | 0.90 | **1.00** | 0.98 |
| #Merkmale 0,5 | 0.99 | 0.93 | 0.99 | 0.99 | 0.99 | 0.96 | 0.99 | 0.99 | **1.00** | 0.96 |
| #Merkmale 0,25 | 0.99 | 0.90 | **1.00** | 0.77 | 0.99 | 0.83 | **1.00** | 0.78 | **1.00** | 0.95 |

Tabelle 6.12.: Ziel Gesundheit, Precision und Recall auf der Testmenge mit „gekauft“= 0

# 7. Fazit

Diese Arbeit wurde mit Unterstützung des Versicherungsunternehmen Allianz Elementar AG Österreich durchgeführt. Dabei wurden personenbezogene Daten genutzt, um eine Vorhersage zu treffen, welcher der Bestandskunden mit mindestens einem aufrechten Vertrag einen zusätzlichen Vertrag abschließen wird. Zur Lösung eines solchen binären Klassifizierungsproblems, wurden verschiedene Machine-Learning-Methoden auf fünf verschiedenen Szenarien mittels Merkmalsauswahl getestet. Das Hauptziel ist, das bestmögliche Modell für diese Cross-Selling Aufgabe zu finden.

Als Erstes wurde in Kapitel 2 ein Zusammenhang zwischen der formalen Cross-Selling Definition hinzu *Data Mining* und schließlich der Definition eines *Recommender Systems* erstellt. Die Auswahl der Methode so eines Empfehlungsdienstes fiel auf einen modellbasierten *Collaborative Filtering* Ansatz. Als Zielvariable für die Klassizifzierung wurden verschiedene Versicherungsrisken auf eine größere Gruppe zusammengesetzt. Dieser Ansatz hat, im Vergleich zu den herkömmlichen Herangehensweisen Produkte für Kunden zu empfehlen, den Vorteil auch ältere und zukünftige Produkte als Trainingsdaten korrekt zuzuordnen. Dadurch sinkt die Anzahl der verschiedenen Ziele aber zugleich steigt auch das Klassengleichgewicht, was sich schlussendlich als entscheidender Faktor herausstellte.

Die dafür ausgewählten Modelle reichen von naiven Klassifizierer über Baummodelle bis zu komplexen Netzwerken: Naive Bayes Klassifizierer, Logistische Regression, Lineare und Quadratische Diskriminanten Analysis, Decision Trees, Random Forest, Gradient Boosting, Neuronale Netze. Aus dieser Auswahl kann ein Vergleich zwischen Modellen mit sehr unterschiedlichen Annahmen gezogen werden. Die Annahmen aller verwendeten Methoden wurden in Kapitel 3 näher beschrieben.

Der Naive Bayes Klassifikator stellt die Annahme, dass die einzelnen Merkmale des zugrundeliegenden Datensatzes bedingt unabhängig voneinander sind. Bei der Logistischen Regression wird davon ausgegangen, dass die Merkmale in einer linearen Beziehung zur Logit-Transformation der Zielvariable stehen müssen. Die LDA Methode zielt darauf ab, eine Trennlinie (oder Kurve im QDA Fall) zu finden, die die Merkmale am besten unterscheidet. Die Annahmen für LDA sind dabei, dass alle Merkmale untereinander unabhängig sind und dieselbe Varianz besitzen. QDA geht davon wie bei LDA davon aus, dass die Merkmale untereinander unabhängig sind und die Kovarianz-Matrix der Merkmale dieselbe ist. Die NB und LR Methode dienen beim Vergleich der Resultate lediglich einem Vergleichswert und deshalb wurden dazu auch keine verschiedenen Hyperparameter getestet. Bei der LDA Methode wurden zwischen einer *singular value decomposition* und einem *least square solution* unterschieden. Bei der QDA wurden verschiedene Regulationsparameter getestet.

Um zusätzlich auch Methoden zu vergleichen, welche keine einschneidenden Annahmen des Merkmalraumes treffen müssen, wurden einige Baum-Strukturmodelle beschrieben. Die Hauptannahme bei diesen Methoden ist, dass sie die Merkmale zuerst auswählen müssen um den bestmöglichen Split im Merkmalraum zu finden. Dieser Prozess wird anschließend

beliebig oft ausgeführt bis eine gewisse Grenze erreicht wird. Es wurden von einfachen Decision Trees weiter zu *bagging* Methoden bis hinzu *Gradient Descent* Algorithmen beschrieben. Damit sollte gezeigt werden ob für die Klassifizierungsaufgabe in dieser Arbeit mehrere *weak learner* ein besseres Resultat liefern als wie klassische Decision Trees. Es wurden für jede Methode eine unterschiedliche Anzahl an maximaler Baumtiefe ausgewählt um *overfitting* zu unterdrücken und um das optimale zu finden.

Als letzte Methode wurde eine Variante der künstlichen neuronalen Netze vorgestellt: Das ausgewählte Netzwerk für auf ein *Feedforward Neural Network* mit mehreren *Hidden Layer*. Bei einem solchen Netz fließt die Information nur in Richtung vorwärts. Dafür wurden die Neuronen mit einer ReLU Aktivierungsfunktion verknüpft und einer Cross-Entropy Kostenfunktion unterstellt. Mit Hilfe der *Backpropagation* kann damit ein Optimierungsproblem formuliert werden und eine optimale Lösung für das gegebene Klassifizierungsproblem gefunden werden. Dabei wurde das NN mit einer unterschiedlichen Anzahl der *Hidden Layer* und Neuronen sowie verschiedenen Regularisationsparameter trainiert um das bestmögliche unter diesen zu finden.

Das Resultat, als Beispiel gezeigt an dem Catboost Algorithmus für das Ziel KFZ, wurde zunächst als ROC und Precision-Recall Kurve gezeigt. Daraus ließ sich bereits ablesen, dass im Grunde alle Methoden einen guten *fit* auf die gegebene Datenmenge liefern. Das Szenario mit PCA Auswahl konnte bereits durch den ROC Graphen als schlechtestes unter allen fünf erkannt werden. Der Blick auf die Precision-Recall Kurve, auch hier stellvertretend (ohne Verlust auf Allgemeinheit) Catboost, zeigt einen großen Unterschied zwischen den Zielvariablen KFZ, Sach und Gesundheit. Um die Ursache dieses Unterschieds genauer begründen zu können, wurden des weiteren die F1-Scores sowie die Precision und Recall Werte auf der gesamten Testmenge sowie auf der Menge für Klasse 1 resp. 2 angeführt. Damit konnte gezeigt werden, dass die Baummodelle bei KFZ und Sach am besten abschneiden. Die Neuronalen Netze zeigten eine Stabilität unter den verschiedenen Szenarien auf und konnten bei der schwersten Klassifizierungsaufgabe, nämlich jene mit dem Gesundheits Risikoprofil, den höchsten F1-Score erzielen. Der NB und QDA schnitten bei allen Szenarien am schlechtesten ab. Eine Eigenschaft was sich bei jedem der Methoden und allen Zielen bemerkbar gemacht hatte, war ein niedriger Precision Score auf der Testmenge mit Klasse 1, im Vergleich zu dem Recall Wert. Und ein umgekehrtes Verhalten auf der Testmenge mit Klasse 0.

Die Analyse des Modells zeigt somit, dass es unterschiedliche Strategien bei der Vorhersage der beiden Klassen verfolgt. Für Klasse 1 weisen alle Modelle eine vorsichtige Prognose auf, was sich in einer hohen Precision und einem niedrigen Recall äußert. Dies bedeutet, dass das Modell bestrebt ist, die Anzahl der falsch Positiv zu minimieren, indem es Vorhersagen der Klasse 1 nur dann macht, wenn es sich relativ sicher ist. Dies führt jedoch dazu, dass viele tatsächliche Instanzen der Klasse 1 nicht erkannt werden, was eine unzureichende Sensitivität zur Folge hat.

Für die Klasse mit „gekauft“= 0 zeigt das Modell eine weniger restriktive Prognosestrategie, wodurch eine hoher Recall erreicht wird, die mit einer geringeren Precision einhergeht. Dadurch wird eine höhere Anzahl von falsch Positiv getroffen, da das Modell dazu neigt, häufiger die Klasse 0 vorherzusagen, selbst wenn diese Vorhersagen teilweise falsch sind.

Um aus der Erkenntnis der Resultate das beste Modell für das Cross-Selling Klassifizierungsproblem auszuwählen, soll deshalb jenes gewählt werden, welches den höchsten

F1-Score auf der gesamten Testmenge erzielt. Somit soll garantiert sein, dass ein Modell mit höherem Gleichgewicht zwischen Precision und Recall auf Klasse 0 und 1 als bestes ausgewählt wird. Dabei soll es erlaubt sein unterschiedliche Modelle für die verschiedenen Risikoprofile auszuwählen. Für KFZ erzielten die Random Forest sowie der Catboost den höchsten F1-Score und Bei Sach erzielte die LightGBM Methode den besten Wert. Die Modelle für das Gesundheit Risikosegment sollten wegen des extremen Klassenungleichgewichts nicht herangezogen werden, nichtsdestotrotz hätte hier das Neuronale Netz den höchsten Score erzielt.

# A. Beweise

### A.0.1. Varianz für B i.i.d. Trees

Seien $X_i$ mit $1 \leqslant i \leqslant B$ die verschiedenen i.i.d. Trees mit $Var(X_i) = \sigma^2$ dann gilt:

$$\begin{aligned} Cov(X_i, X_j) &= 0 \\ Y &= \frac{1}{B}\sum_{i=1}^{B} X_i \\ Var(Y) &= \frac{1}{B^2}\sum_{i=1}^{B} Var(X_i) \\ &= \frac{1}{B}\sigma^2 \end{aligned}$$

### A.0.2. Varianz für B i.d. Trees

Seien $X_i$ mit $1 \leqslant i \leqslant B$ die verschiedenen i.d. Trees mit $Var(X_i) = \sigma^2$ und Korrelationskoeffizienten $\rho > 0$ dann gilt:

$$\begin{aligned} Cov(X_i, X_j) &= \rho\sqrt{Var(X_i)}\sqrt{Var(X_j)} = \rho\sigma^2 \\ Y &= \frac{1}{B}\sum_{i=1}^{B} X_i \\ Var(Y) &= \frac{1}{B^2}\sum_{i=1}^{B} Var(X_i) + \frac{1}{B^2}\sum_{i\neq j} Cov(X_i, X_j) \\ &= \frac{1}{B^2}\sum_{i=1}^{B} \sigma^2 + \frac{1}{B^2}\sum_{i\neq j} \rho\sigma^2 \\ &= \frac{1}{B}\sigma^2 + \frac{1}{B^2}(B^2 - B)\rho\sigma^2 \\ &= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \end{aligned}$$

# Literaturverzeichnis


[1] Wagner A. Kamakura. Cross-selling. *Journal of Relationship Marketing*, volume 6(number 3-4):pages 41–58, 2008.

[2] Deepjyoti Roy and Mala Dutta. A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9, 2022.

[3] Grand View Research. Customer relationship management market size, share, & trends analysis report, by component, by solution, by deployment, by enterprise size, by end use, and segment forecasts, 2024 - 2030. `https://www.grandviewresearch.com/industry-analysis/customer-relationship-management-crm-market/`. Aufgerufen am 26.05.2024.

[4] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, volume 17(number 3):page 37, 1996.

[5] Yunus Emre Özdemir and Selim Bayraklı. A case study on building a cross-selling model through machine learning in the insurance industry. *Avrupa Bilim ve Teknoloji Dergisi*, (number 35):pages 364–372, 2022.

[6] Tomas Sidorowicz, Pedro Peres, and Yuhua Li. A novel approach for cross-selling insurance products using positive unlabelled learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022.

[7] Ali Galip Sekeroglu. Impacts of feature selection techniques in machine learning algorithms for cross selling: A comprehensive study for insurance industry. 2021. Aufgerufen am 12.11.2024.

[8] Maleeha Qazi, Glenn M. Fung, Katie J. Meissner, and Eduardo R. Fontes. An insurance recommendation system using bayesian networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, page 274–278. Association for Computing Machinery, 2017.

[9] Lior Rokach, Guy Shani, Bracha Shapira, Eyal Chapnik, and Gali Siboni. Recommending insurance riders. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, page 253–260. Association for Computing Machinery, 2013.

[10] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, volume 2009:Article ID 421425, 2009.

[11] Harry Zhang. The optimality of naive bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004*, volume 2, 2004.

[12] M. Narasimha Murty and V. Susheela Devi. *Pattern Recognition - An Algorithmic Approach.* Undergraduate Topics in Computer Science. Springer, 2011.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., 2001.

[14] Robert Messenger and Lewis Mandell. A modal search technique for predictive nominal scale multivariate analysis. *Journal of the American Statistical Association*, volume 67(number 340):pages 768–772, 1972.

[15] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 278–282. IEEE Computer Society, 1995.

[16] Leo Breiman. Random forests. *Machine Learning*, 45(number 1):pages 5–32, 2001.

[17] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, volume 2(number 3):pages 18–22, 2002.

[18] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, EuroCOLT '95, page 23–37, Berlin, Heidelberg, 1995. Springer-Verlag.

[19] Trevor J. Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and Its Interface*, volume 2:pages 349–360, 2009.

[20] Harris Drucker and Corinna Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems*, volume 8, pages 479–485, 1995.

[21] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, volume 29:pages 1189–1232, 2000.

[22] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting algorithms as gradient descent. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[23] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, volume 38(number 4):pages 367–378, 2002. Nonlinear Methods and Data Mining.

[24] Machine learning challenge winning solutions. `https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions`. Accessed: 2024-05-26.

[25] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794. Association for Computing Machinery, 2016.

[26] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157. Curran Associates Inc., 2017.

[27] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 1st edition, 2012.

[28] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[29] Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, volume 5:pages 127–147, 1943.

[30] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, volume 65(number 6):pages 386–408, 1958.

[31] Henry J. Kelley. Gradient theory of optimal flight paths. *ARS Journal*, volume 30:pages 947–954, 1960.

[32] Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, volume 5(number 1):pages 30–45, 1962.

[33] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, volume 323(number 6088):pages 533–536, 1986.

[34] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[35] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Journal of Machine Learning Research*, volume 15, 01 2010.

[36] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, volume 2(number 3):pages 183–192, 1989.

[37] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, volume 2(number 5):pages 359–366, 1989.

[38] Theodor Van Veen. The neural network zoo. `https://www.asimovinstitute.org/neural-network-zoo/`. Accessed: 2024-05-26.

[39] XuGang Wang, Tang Zheng, Hiroki Tamura, Masahiro Ishii, and W. Sun. An improved backpropagation algorithm to avoid the local minima problem. *Neurocomputing*, volume 56:pages 455–460, 2004.

[40] Alvaro H.C. Correia, Daniel E. Worrall, and Roberto Bondesan. Neural simulated annealing. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 4946–4962. PMLR, 25–27 Apr 2023.

[41] Rahul Yadav, Swapnil Tripathi, Shailendra Asati, and Malay Das. A combined neural network and simulated annealing based inverse technique to optimize the heat source control parameters in heat treatment furnaces. *Inverse Problems in Science and Engineering*, volume 28:pages 1–22, 2020.

[42] L.M. Rere, Mohamad Ivan Fanany, and Aniati Arymurthy. Simulated annealing algorithm for deep learning. In *Procedia Computer Science*, volume 72, 2015.

[43] Yann L. Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. San Francisco, CA: Morgan Kaufmann, 1990.

[44] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

[45] Mamdouh Refaat. *Data Preparation for Data Mining Using SAS*. Morgan Kaufmann Publishers Inc., 2006.

[46] Xindong Wu. *Knowledge acquisition from databases*. Ablex Publishing Corp., USA, 1996.

[47] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22:177–210, 2003.

[48] Mustafa Alabadla, Fatimah Sidi, Iskandar Ishak, Hamidah Ibrahim, Lilly Affendey, Zafienas Che Ani, Marzanah Jabar, Umar Bukar, Navin Kumar Devaraj, Ahmad Muda, Anas Tharek, Noritah Omar, and Izham Jaya. Systematic review of using machine learning in imputing missing values. *IEEE Access*, 10:1–1, 2022.

[49] Nearest neighbor imputation. `https://scikit-learn.org/stable/modules/impute.html#knnimpute`. Accessed: 2024-07-11.

[50] Shichao Zhang. Nearest neighbor selection for iteratively knn imputation. *J. Syst. Softw.*, 85(11):2541–2552, nov 2012.

[51] G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[52] In-Kwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

[53] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer-Verlag, 2002.

[54] Jonathon Shlens. A tutorial on principal component analysis. *CoRR*, abs/1404.1100, 2014.

[55] Lindsay I Smith. A tutorial on principal components analysis. Technical report, Cornell University, USA, 2002.

[56] Jianwei Zheng and Cyril Rakovski. On the application of principal component analysis to classification problems. *Data Science Journal*, 20, 08 2021.

[57] Rangachari Anand, Kishan G Mehrotra, Chilukuri K Mohan, and Sanjay Ranka. An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969, 1993.

[58] Miroslav Kubat, Robert C. Holte, and Stan Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2-3):195–215, 1998.

[59] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, 2002.

[60] Alberto Fernández, Salvador García, Francisco Herrera, and Nitesh V. Chawla. Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *J. Artif. Int. Res.*, 61(1):863–905, 2018.

[61] Nearest neighbor imputation. `https://www.analyticsvidhya.com/blog/2023/11/train-test-validation-split/`. Accessed: 2024-07-18.

[62] Scikit-learn select k-best dokumentation. `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html`. Accessed: 2024-11-06.

[63] Tom Fawcett and Foster Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1:291–316, 1997.

# Abbildungsverzeichnis

# Tabellenverzeichnis